A remedy perspective for emerging antipatterns in agile software projects in small and medium enterprises

Elena Soloveva

University of Tampere
Faculty of Natural Sciences
Degree Programme in Software Development
Elena Soloveva: A remedy perspective for emerging antipatterns in agile software projects in small and medium enterprises
Supervisors: Eleni Berki, Juri Valtanen
M. Sc. Thesis Proposal, 63 pages, 5 index and appendix pages
December 2017

University of Tampere
Faculty of Natural Sciences
Degree Programme in Software Development
Elena Soloveva: A remedy perspective for emerging antipatterns in agile software projects in small and medium enterprises
M.Sc. thesis, 63 pages, 5 index and appendix pages
December 2017

Agile manifesto transforms mindset of software development teams, with the purpose of enabling them to provide rapid deliveries. Lack of conceptual clarity in the agile theory leads to multi-faceted agile nature, with teams interpreting and implementing agile values variously. Team decisions present paramount importance in the agile setting, with poor decision making quality being a significant factor in causing failures of the software development projects.

An antipattern emerges when a conventional solution generates negative consequences. The agile manifesto provides theory, which in practice may result into problems, thus creating antipatterns. In this research, implications of the agile manifesto were outlined based on its critical analysis, with focus on decision making as a pivotal challenge in the agile usage. Furthermore, empirical study revealed obstacles in the agile manifesto implementation. Consequently, collaboration, short-term thinking and lack of planning issues were found to be the most influencing decision making process.

While agile methodology advocates continuous process and product improving, the conducted case study illustrates that decision making issues are hard to reveal by teams as they are restrained by their mindset state and tend to stick to decisions from the past experience. The refactoring for the decision making management in the agile software development has been developed based on a) the knowledge of quality issues in decision making, b) the software project management antipatterns, and c) “Theory W” paradigm for the effective project management.

Understanding of agile manifesto challenges and quality issues in the agile usage is indispensable to increasing the projects’ success. Addressing the identified issues results in agile software development process quality improvement.

Key words and terms: agile, management antipatterns, decision making, software quality management, software process improvement, agile manifesto challenges, agility.

Acknowledgements

I am sincerely grateful for the scientific support and encouragement I received during this thesis work. First of all, I would like to express my gratitude to the supervisors, *Eleni Berki* and *Juri Valtanen*. The consultancy given in the questionnaire construction, review of intermediate results and guidance was especially helpful in moving on and learning the research principles.

This work would not be possible without assistance of the company I am working in, it is a real pleasure to rub shoulders with them developing the software.

I want to thank my family, friends and especially *Rike Leppala* for always being considerate to me in the endeavor to finalise the work within limited time.

Contents

1. Introduction

In order to keep up with the demands for constant software evolutions, there is a need to react to and adopt changing software requirements fast. Furthermore, with the necessity of deploying software online, such as cloud or web applications and services, the competition on the market has further increased, calling for rapid and effective software development [Zhu et al. 2016]. The capability of an information systems development (ISD) method to manage changes swiftly and "learn from change while contributing to perceived customer value (economy, quality and simplicity)" defines its agility [Conboy 2009].

Agile methods appear to address agility, yet, they hold severe conceptual shortcomings, making it hard to apply and measure agility. Conboy [2009] concluded that agility is an imprecise and multi-faceted concept, given that it comprises a plenty of different methods and is highly exposed to diverse interpretation and application. Moreover, agility is polymorphous and multidimensional.

An extent of achieved agility should be assessed within the context in which an agile method is applied [Alleman 2002]. Each project is unique, and in this regard, it adopts agile values and practices in its own way, depending on the team's understanding of agile methodology, consideration for the needed set of practices and approaches of how to apply them.

1.1. Decision making role in achieving agility

There is a lack of rigorous guidelines describing how to implement agile software development in the projects to achieve fast deliveries [Conboy 2009]. Moreover, the agile manifesto sets challenges for the software development teams, thus moving from agile theory to practice is not straightforward. As a result, it is difficult for the teams to achieve agility, and it is important to determine key contributors defining success in the undertaken endeavour. Communication, *collaborative decision making* and iterative development processes are considered to be critical when dealing with changes and can be defined as the prominent determinants of the extent of agility [Rathor et al. 2016].

Software product is not a result of single individual work, but the outcome of collaborative team contributions. Traditional methods emphasized focus on achievements of each individual, creating competitive atmosphere, rather than moving to the project goal as a collaborative team [Hackman 1987]. In contrast, in agile

software development the team takes the central role and is given the ownership for the software development decisions [Agile Alliance 2001].

The chosen process of software development and selected methods highly contribute to the quality of the resulting product [Carleton et al. 1994; Acton et al. 2014]. However, software development is a social phenomenon, with people still being the most influencing software engineering factor [Cockburn 2001]. With an emergence of agile software methodologies, the original focus in software engineering has moved from process to people determining the project success [Broza 2012]. Since the focus in defining methodologies was on process rather than its use by practitioners, there is an evident gap as this approach neglects interpretation by people and human factor [AlQaisi et al. 2017].

Research on quality issues in agile software development is limited, with few studies looking into concerns of quality in agile methodology and highlighting the difficulties to apply metrics to soft or uncontrollable factors prevailing in agile software development, for instance knowledge creation and transfer [Berki et al. 2007]. In this thesis, the quality of agile software development is evaluated from the extent to which the teams are successful in:

- Interpreting the core values of the agile manifesto.
- Applying approaches, chosen to satisfy agile manifesto's ambitions, such as achieving team collaboration and team decisions quality on numerous factors arising throughout the software development process.

The project success strongly relies on the ability of the team to identify quality issues and implement corrective actions before they turn into project failures to support dynamic software evolution demands [Laplante and Neill 2006]. Understanding the obstacles that the agile team may face when using agile methodology is instrumental for practitioners to reveal and address corresponding issues. Moreover, addressing the issues linked to the agile manifesto implementation, could be achieved by applying solutions of antipatterns, being one of the underlying reasons of the problems.

While patterns are created as a generic solution to recurring issues, antipatterns describe solutions which relate to bad practices. Such solutions are considered to bring advantages at the first glance and thus appealing to implement, but when applied in a real case scenario, result in negative consequences on the project. Since antipatterns are one of the issues' causes, the problems resolution can be achieved by applying solutions to the related antipatterns. [Aydinli et al. 2016]

There are few studies investigating antipatterns in agile software development, many focus on the agile practices usage [Eloranta et al. 2013; Kral and Zemlicka 2007; Kuranuki and Hiranabe 2004] rather than the agile manifesto core values implementation itself. However, software project management antipatterns described by Laplante and Neill [2006] look into the mindset at individual and team level, and thus can be useful to address antipatterns in the agile manifesto usage.

1.2. Research goal and questions

The overall aim of this study is to identify quality issues in agile software development, with focus on decision making as a pivotal challenge when applying agile methodology in practice. In particular, the research will investigate:

- Influence of agile manifesto on decision making process.
- Characteristics of decision making in technical meetings.
- Influence of issues in a decision making process on quality of user stories and tasks, architecture design, delivery infrastructure support, code inspection and tests.
- Software project management antipatterns for addressing the identified issues, thus improving agile software development process.

In order to achieve the study objectives, the research aims to answer the following questions:

*Research Question 1***:** What are the challenges agile software development methodology sets for teams?

1.1 What kind of implications the agile manifesto statements present for the team?

1.2 How do agile values influence decision making in an agile team?

*Research Question 2***:** What are the quality issues agile teams may face when implementing the core values of the agile manifesto? (Case Study)

2.1 What are the quality issues in the team's decision making process?

2.2 What are the other quality issues and what is their correlation in regard to decision making?

*Research Question 3***:** What could be the agile software development process quality improvement?

3.1 Which techniques could be proposed for agile teams to identify decision making issues?

3.2 How quality issues, could be addressed based on the software project management antipatterns?

1.3. Research outline

The research is focused on collaboration and decision making, primarily in the technical meetings. Case study is limited to the analysis of data collected based on the survey taken by a single team in Finland. While it does not reveal all potential issues teams can face when implementing agile, the issues identified in the agile manifesto implementation raise awareness of presence of antipatterns in the agile usage and the quality risks agile teams take. Furthermore, when the issues are identified, it is vital for the teams to be aware of how to address them. Since the problems found are linked to the agile manifesto core values, this indicates that issues are related to the antipatterns in the agile usage. Thus, software project management antipatterns are considered to be useful to address the issues, and are consulted for providing a remedy.

Further on, the paper is divided into six main chapters. The Chapter 2 provides background, introducing agile software development and software quality definitions, highlighting their multi-faceted nature and importance of context in the software quality evaluations. The thesis proceeds then into critical analysis of the agile manifesto and focuses on decision making as a pivotal challenge in agile setting, with decisions eventually determining the project success. The Chapter 4 outlines survey-based case study identifying quality issues in the decision making process and resulting decisions, with the results presented in Chapter 5. Finally, the quality issues are discussed and a remedy is suggested in the Chapter 6, with conclusions summarized in the Chapter 7.

2. Background

2.1. Agile Software Development

Agile software development was originated with a definition of its core values in the agile manifesto, introduced by Cockburn and other practitioners [Agile Alliance 2001]:

> *"Individuals and interactions over processes and tools.*
> *Working software over comprehensive documentation.*
> *Customer collaboration over contract negotiation.*
> *Responding to change over following a plan. "*

Agile development model advocates a lightweight approach to software development in face of demands for fast software changes. This innovation in ISD was also promoted by Cockburn et al. [2001] as an advantageous software development. It can be noted that agile methodology is in favor of simplicity and eliminates unnecessary activities by setting the agile manifesto's values which should drive the decisions of agile teams.

Agile is a methodology which encapsulates a set of different methods. In spite of each of them describe a unique approach, they all inherit the values declared by the agile manifesto. In contrast to the predictive methods of software development which are foundation to traditional software development methods (such as waterfall), agile software development is adaptive. Traditional methods rely on availability of detailed requirements analysis and planning in the beginning of the software development, while agile methods are based on iterative model. Each iteration presents "small waterfall", involving requirements analysis, design, implementation and verification. Importantly, it implies that a result of an iteration is a working software targeting only requirements included in the iteration. The development cycles are continuous, incrementally adding features in each iteration, until the complete product is released. [Stoica et al. 2013]

Software development can be highly challenging and in that case need more sophisticated and diverse solutions that are combination of various methods and approaches [Ambler 2013]. Thus, there is no silver bullet to succeed in the software development, set of practices used in the projects varies in order to address the problems as to the project needs and the context.

The companies have to decide which agile software development method to select to speed up deliveries in their projects contexts. It makes it hard because there are

plenty of methods which effectiveness when integrated depends on various factors, such as the organizational structure, culture and relationship with the customer. There are multiple studies that aim at improving understanding of facets significantly influencing the choice. [Chow and Cao 2008; Ambler 2009; Misra et al. 2009]

Most of the companies implement Scrum, Kanban or XP. The methods when put into the projects context are never the same compared to the ones in the textbooks format, and many companies use hybrid approaches. [Drury et al. 2017]

2.2. Software quality

2.2.1. Diversity in views on quality definition

Quality is a multifaceted term, which can be defined in a number of ways. The Institute for Electrical and Electronic Engineers (IEEE) [1990] introduces quality as "the degree to which a system, component, or process meets specified requirements or customer or user needs or expectations". In an attempt to provide a quantitative view of software quality, the definition has further evolved to "the degree to which software possesses a desired combination of quality attributes" [IEEE Computer Society 2004].

Garvin [1984] suggested that quality can be defined from numerous aspects:

Transcendent-based definition

The transcendent approach holds that quality is something that cannot be defined accurately. Understanding of quality is only achievable through experience.

Product-based definition

The product-based approach states that quality is measurable through a set of attributes. Thus quality level is defined by these variables values.

User-based definition

The user-based approach suggests that quality is subjective, and is determined by the users of the product based on their perspective.

Manufacturing-based definition

The manufacturing-based approach equates quality to compliance to requirements: if a product is built as specified, it exhibits quality.

Value-based definition

The value-based approach holds that a quality product is a product that provides "performance at an acceptable price or conformance at an acceptable cost" [Garvin, 1987].

2.2.2. Software engineering process quality for software quality

Software quality through functional correctness

There are multiple insights gained into software quality based on the quality definitions. The ISO25000 standards, as well as quality models by Boehm, Hyatt and Rosenberg and McCall et al. utilize metrics for measuring software quality. Duvall [2007] defined the quality as a "measurable specification just like any other". The researcher attributes to the quality metrics, including maintainability, performance, security, extensibility and readability.

Moreover, software quality is often measured based on testing results for the product requirements. The testing process reveals bugs that need to be fixed to provide the quality. However, it is not the matter of one software development activity, such as test cases execution. If the executed tests are passed, it does not necessarily indicate excellent product quality. The quality statement becomes valid only if test cases are of high quality. Test coverage together with tests review can provide an extent of achieved test cases quality. The coverage can be then measured through requirements specification coverage, test inputs range coverage, code coverage to see which parts were covered by tests. It becomes evident, the set of activities are involved in determining the quality. [Duvall et al. 2007]

Software quality through software development process excellence

Quality is not only about functional correctness, it is directly related to the quality of the software development process. The software engineering process describes a life cycle of software development. It consists of numerous activities, leading to the final software product.

Software industry standards, such as the ISO9000, CMMI and the Personal Software Process (PSP) focus on processes excellence providing a set of "best practices" for software development [Acton et al. 2014].

In order to validate software development quality, defects found late can serve as a signifier for software development process improvements need. Secondly, it includes a feedback loop on software changes. [Morris 2016]

Software quality through agility of software change

"Developing software requires planning for change, continuously observing the results, and incrementally course correcting based on the results. " [Duvall et al. 2007]

The longer issues stay undetected is an indicator of the poor system quality. The issues, when found late in the software development process, is harder to localize and fix. In that case the feedback on the changes is deferred, thus delaying the system change.

Poor software quality makes the changes difficult to handle. For instance, when the software code is hard to understand for a developer who needs to incorporate a change in the system, it takes more efforts to modify it for a new use case. The extra time is spent to grasp the current functional logic. The code complexity increases likelihood of breaking legacy use cases or introducing new unplanned behavior scenarios. In that case the poor code quality causes even a simple change to be costly. [Morris 2016]

So, if we identify factors interfering into applying rapid changes to the software, then we can improve the quality. The pace of how quickly and safely the change can be applied to the system can be used as a measurement of the quality. Consequently, the quality measure calls to a question whether it is easy and fast to safely implement a new change to the system.

The quality is difficult to achieve and maintain without well integrated software engineering practices which provide continuous feedback. In spite of most organizations use continuous integration and deployment, they still do not gain all power of them. It requires additional efforts to learn using them efficiently. [Duvall et al. 2007]

2.3. Collaboration between software operations and development teams

Projects developing online products, such as web and mobile applications, software as a service, etc., are forced to deliver changes swiftly to sustain the end users online activities. An update of the system may cause an impact on the service behavior in the production, affecting the users' experiences. The system nonfunctional requirements, such as reliability and performance are of great concern when deploying the software to the production environment. Deployment operations is an integral part of the service delivery, which impacts the change along with development. [Zhu et al. 2016]

In case in the organization different teams were responsible for software development and its deployment, there was a need of the constant information flow between them to communicate the release content and provide feedback from the system reflecting response to the change in the production environment. This handshaking was adding extra lead time to the deliveries and was a subject to multiple

communication issues. Operations (production) team knowledge was concentrated in the domain of infrastructure, involving build, testing and deployment. In case of the software related issues manifested after release, developers had to be involved for troubleshooting. Moreover, the development team had conflicting goal to the quality assurance/production teams. Software engineers were driven by idea of making deliveries swiftly and often, while operations and quality team preference was rather to have more predictable changes, meaning well planned releases with detailed documentation and more extensive integration verification scopes. With agile software development incorporation in the companies, the need of efficient and fast deliveries increased. [Zhu et al. 2016]

Agile movement encouraged collaboration between different groups, creating cross-functional teams, shared workspace and shifting the focus on working software rather than on the process and documentation. Improved cooperation made software releases more efficient. [Zhu et al. 2016]

The operations team could now learn and incorporate development best practices into build, test and deployment preceding the release, including revision control tools, use of design patterns, extreme programming, and continuous integration. And vice versa, the development team could now better understand operations environment and take end-to-end responsibility for the delivery. They could learn from production team their orientation of practices to service, quality, and reliability; how to perform deployments and configure environment. [Spinellis 2016]

However, the deployment process still required optimizations to support rapid development. The demands of agile deliveries caused a need of new solutions for architecture and deployment process that will make software more reliable, more scalable, and easier to deploy, configure, and run.

3. Literature Review. Decision Making as Agile Usage Challenge

This chapter examines existing research on agile challenges and decision making quality in agile software development. It starts with analysis of agile values statements available in the literature, and then flows into comprehension of decision making models. Applying of agile values into practice greatly relies on abilities of agile teams to make quality decisions, which requires broad range of skills, such as people-oriented (collaboration / communication), brainstorming, planning, continuous integration skills and so on. By looking into decision making process, the impact on decision quality can be clarified [Dewan and Hansen 1994]. This, in turn, is crucial, as poor decisions possess a software quality risk, including incomplete features, unexpected defects, unreliable estimates [Drury, Conboy and Actonb 2017].

Based on the literature review, the survey will be designed in order to get insight into how one of the teams in Finland implements agility values, considering the decision making and quality issues the team members think they have in the project compared to the agile theory.

3.1. Agile manifesto implications

The teams' decisions present paramount importance in the agile setting. Agile manifesto intends to transform the team mindset in order for them to incorporate software changes fast. The interpretation and application of the agile value statements into work is under teams considerations. The manifesto indicates what the highest priorities are when there is a need to make a trade-off to support a quick change. For example, "individuals and interactions are over processes and tools" [Agile Alliance 2001]. Since it is not a concrete solution but rather the suggested way of thinking, there is freedom in understanding the proportions of each compound when it comes to practice. For example, to which extent planning should be reduced, or to which extent processes are needed. Hence, the four key agile values present challenges in the decision making [Drury et al. 2017]. To aid developers in grasping the values captured in the agile manifesto, the Agile Alliance [2001] clarified them into 12 principles. The thesis will focus on reviewing these principles in order to reveal changes and challenges in regards to agile team's decision making process.

3.1.1. Prioritising importance of individuals and communication

The first agile core value is “Individuals and Interactions over processes and tools” [Agile Alliance 2001]. The following agile principles relate to a teamwork and it can be noticed, that there is a strong highlight of the *self-organizing and self-improving* as a team essential properties:

“*The best architectures, requirements, and designs emerge from self-organizing teams. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

Build projects around motivated individuals.

Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.”

The statements convey that team work is paramount in agile settings and the team should be able to:

1. *Self-organize.* The team is gradually learning and gaining management and technical skills to take decisions, which previously were outside of traditional developers skills. It can create situation where the team does not yet have skills or knowledge for effective decision taking. It is also unclear to which extent teams are looking for the information and skills development when taking decisions. [Drury et al. 2017]
2. *Effectively collaborate.* Knowledge sharing on the system (UI development, architecture, testing, infrastructure, etc.) occurs through high *team collaboration*. There is no specified roles, the knowledge is not narrowly concentrated, but instead shared between everyone in the team and roles are merged. At the same time, expertise in different areas feeds finding optimal decision [Hackman 1987; Seers et al. 1995].
3. *Improve work.* Daily collaboration, and retrospective meeting in the end of the iteration serve the purpose of reflection on the team work and identification of improvements. However, people relate causes to what they see rather then what is operating behind the hood [Hackman 1987]. The issues in decision making may be hard to reveal and thus improve. The teams also tend to take repetitive decisions, based on the previous experience [Drury et al. 2017]. Thus, the team may stick to the previous choices instead of analyzing the situation. This also means the team

may be on the course of certain way of thinking all along the process, not being able to realize weak points and thus improve.

3.1.2. Prioritising importance of collaboration with customer

The second agile core value is “Customer Collaboration over contract negotiation” [Agile Alliance 2001]. The following agile manifesto principles relate to interactions with customer [Agile Alliance 2001]:

“Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Business people and developers must work together daily throughout the project.”

Delivery of valuable software assumes high customer motivation in the development of the product [Chow and Cao 2008], as well as being knowledgeable about the domain [Qasaimeh et al. 2008]. In addition, according to Qasaimeh study, the approach to customer collaboration varies between agile methods. Depending on the degree of customer involvement and competence in the domain area, the projects may choose the most suitable agile methodology practice [Kasem and Razali 2017].

In case customer is active and have some vision on the product, then it is easier to define what the next expected valuable solutions are and receive constructive feedback. Not only the customer, but the team can suggest possible deliverables, and align the content and schedule with the customer. This is the concept of collaboration, and it is encouraged to be continuous to provide correct set of functionalities. [Kasem and Razali 2017]

Insufficient or not effective customer involvement can undermine collaborative atmosphere [Coffin 2016]. At the extreme side, there are projects where the team has to take complete ownership to identify the deliverables by themselves. Then the team is highly responsible to determine what is valuable and prioritize work accordingly. It showcases that ability for the team to identify what user stories are the most valuable is vital and for some projects is crucial.

3.1.3. Prioritising importance of fast acting on changes

The third agile core value is “Responding to Change over following a plan” [Agile Alliance 2001]. The following agile principle regards the changes handling, specifically addressing the case when they interrupt planned activities: *“Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.”* [Agile Alliance 2001]

Software development methods can vary from being adaptive to, opposite, predictive. The approach to planning in the project is one of the key characteristics influenced by this factor. Agile, being adaptive, gives flexibility in how to achieve goals, while predictive software development methods focus on detailed planning. Agile teams may not know which tasks they will do next week, more prevalent is instead having a clear closest milestone to achieve [Cockburn 2001].

The agile manifesto suggests "Responding to Change over following a plan". This puts teams' mindset towards making everything to facilitate fast adaptivity to a change, which means less strict planning. The extent of planning reduction should be defined keeping in mind other agile development principles, like continuous improvement, good design and architecture.

Since multiple variables are involved in this equation, teams' decisions on what minimal planning varies in the projects.

3.1.4. Prioritising importance of working software

The third agile core value is "Working software over comprehensive documentation" [Agile Alliance 2001]. The following agile principles relate to the documentation handling and importance of frequent deliveries:

"Working software is the primary measure of progress.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
Simplicity--the art of maximizing the amount of work not done--is essential.
Continuous attention to technical excellence and good design enhances agility."

The goal of an agile team is to incorporate agility in the software development process, releasing the product often. The agile principles advocate attention to the "technical excellence" and "good design", which in its turn imply higher cost for solution design and implementation. Hence, this creates equation for the team to find for each technical solution a balance between agility and technical excellence.

Focus on providing higher value implies concentrating only on the use case that brings value right now. This means also while designing the solution, the team may decide that consideration of affected / generated by the solution boundary use cases is secondary. As a result, the team may not reveal impacts on the architecture and its stability in that case. Such technical decisions of the team which favor shortcuts in the

solution design phase may incur technical debt. The term technical debt was initially coined by Ward Cunningham [1992], which refers to omitted quality in the solutions. Buschmann [2011] also identified that not only this can happen unintentionally, but some teams use technical debt to accelerate the development pace. In the conducted by Buschmann case study, this led to major performance issues that turned later on to business consequences.

3.2. Overview of decisions in Agile Software Development

Agile methodology positions teamwork as a collaborative work, replacing the concept of individual work focus which is used in traditional development [Vinekar et al. 2006]. Consequently, agile development in comparison to the traditional software development methods introduces:

- Multidisciplinary vs. specialized skillset.
- High collaboration vs. individual work.
- Pluralist decision making vs. manager as accountable for decisions.

Agile software development teams make series of critical decisions that eventually result in the project success or failure [Conboy 2009]. At the same time, the project managers are less involved as decision makers [Alleman 2002], empowering agile teams to take decisions. On the contrary to the declining role of project manager's influence, customers still greatly impact the product development decisions in agile teams [Beck 2000].

Agile team engages in making of numerous decisions during the software development cycle [Drury et al. 2011]. Agile teams' decision making process exhibits the use of minimum information, reduced analysis and comparing of bad and good aspects of alternatives [McAvoy and Butler 2009]. Studies of Briggs and Reinig, Coyle and other researchers are consulted in the thesis regarding decision making characteristics in an agile context.

Agile team engages in making of numerous decisions during the software development cycle. As to the studies [Drury et al. 2011], team is accountable for broad range of decisions:

Iterations planning

- Discussion of user stories and tasks for the iteration, estimations and identification if user stories need further studies, the need to make user stories smaller.

- Discussion of priorities within the iteration.

Iterations execution

- Discussion of acceptance of the user story as ‘done’.
- Discussion of interfaces design.
- Discussion of the architecture/design solutions.
- Discussion of design solutions for functionality implementation.
- Discussion of which tests are needed.
- Discussion of when commit the code.
- Discussion of need of change.

Iteration Review

- Discussion the customer expectations are met.
- Discussion whether to accept the work done in the iteration.
- Preliminary view on user stories to be included in the next sprint.

Iteration Retrospective

- Discussion of what went well and should be kept up.
- Discussion of what went wrong / not effective, and how to fix it in the next iterations.
- Discussion of improvements for the next iteration.

If planning session is insufficient in generation of clear user stories and solutions, the team starts user stories implementation with higher uncertainties in the solutions design and estimations. This creates the following probable issues with impact on quality:

- Tasks taking longer than 2-3 days to implement (can cause big bang integration and slower feedback).
- Not optimal or incomplete solutions (can cause a need of rework further on).
- Difficulty in breaking down user stories into tasks (can slower down the team work pace).

The teams can reduce these risks impact by organizing separate discussions to clear up uncertainties as soon as the concerned user story implementation is started. It differs from the iteration planning session as involves just several participants instead of the whole team. In this setup the risks compared to the planning meeting involving the whole team:

- This group of team members may not have enough technical knowledge to design best solution for the selected user story.
- The impact of individualities on final decision is higher, increasing the chance of taking solution not objectively.
- This group may lack critical thinking compared to when perspectives of all team members are heard
- Knowledge on the solution is only shared with the meetings participants, other team members will lack system perspective. Missing of this knowledge may adversely complicate taking further decisions and implementations.

The team would probably notice impact on quality when the issue with solution design becomes apparent during code review or later when adding new features on top. The issue with missing knowledge sharing at this moment may reflect in lack of technical argumentations from other team members for the concerned topic. When developers do not know how some parts of the system work, they may have wrong or lack of opinion in further technical discussions.

3.3. Decision making models: from theory to practice

Decision making models are differentiated by the method they rely on to describe the decision making process. Normative models, which aim to reveal the ways leading to the optimal solution, provide comprehensive theory. Furthermore, prescriptive models refine normative models with aspects taken from real case observations, making it easier to apply and compare the models with decision making undertaking in practice. In contrast, descriptive decision making models are looking at decision making from an opposite angle, focusing not on the theory but behaviors observed in a real case scenario. [Dillon 1998]

3.3.1. Rational and prescriptive decision making

Rational decision making (RDM) is seeking to answer the question what people in theory should do in the decision making process to be able to find the optimal outcome for the situation in the context. Firstly, RDM as a normative decision theory [Simon 1979] introduces decision makers as sensible, extremely skilled beings who easily combat their fears and insecurities in offering opinion [Bell et al. 1988]. Secondly, RDM implies decision makers are fully informed, and consequently, are able to

generate complete set of alternatives and then define the most satisfactory solution between existing choices by following linear, sequential steps [Drury et al. 2011]. Prescriptive Decision Making (PDM) is built upon RDM findings but makes further step to specify what people can actually implement from what RDM suggests in theory.

Simon's model

Simon outlined three stages of decision making processes [Simon 1979]. This model created the baseline for further research in decision making process. The phases are the following:

- Intelligence

  This is the first phase, and it relates to the activity of collecting information to specify the need of the decision. It explores the problem environment, with the problem formulated afterwards.

- Design

  This is the second phase, and it refers to the research of the actions to solve the problem, eventually developing a set of alternative options.

- Choice

  This is the final phase, and it is concerned with selecting the appropriate solution among available options.

Each stage can be viewed as a complex decision making activity because each can generate new issues, itself requiring initiation of a separate decision making process. Moreover, Simon's model shows that the process can go in different fashions, either linear or iterative (when the decision makers need to revisit previous phases as to the developed vision).

3.3.2. Descriptive Decision Making

Descriptive decision making (DDM) looks into how decision making applies in real case scenarios [Bell et al. 1988]. In the foundation of DDM lays *Bounded Rationality*, also referred as *Limited Rationality*, which was first introduced by Simon [1979]. The concept explains that rational behavior commences within constrains, involving a spectrum of cognitive factors. These limitations are conceived by the nature of human beings.

In particular, identifying alternatives [Barron 1974] and an optimal option [Lipshitz et al. 2001] does not follow RDM prescriptions, thus decision makers do not necessarily find the best solution. Dillon's [1998] review of existing descriptive theories

emphasised the impact of *Bounded Rationality* on decision making and importance of understanding the decision makers' behaviour, as well as related reasons. Payne et al. [1993], Divekar et al. [2012] and other researchers' works elucidate aspects of human limitations and behaviour significant in reducing rationality of decision makers.

Identifying all alternatives

The outcome of DDM modeling is that the set of generated by decision makers options does not cover all potential, thus in a real life situations prescribed by RDM solution could be initially missing in the produced options scope [Barron 1974].

Choosing an optimal alternative

In turn, Lipshitz advocated that human activity of making best choice can be characterized by at least one of three modes: consequential, matching and reassessment choice. The framework views decision making actions as *argument driven* rather than forward-looking choice which is based on consequences of the decisions relative to the goal. Additionally, when decision makers compare alternatives, this is *not necessarily done in a "systematic way"* [Lipshitz et al. 2001].

With the investigations conducted to explain the decision making in a given situation, it becomes evident that the actual choice can be different from prescribed by influence of multiple factors. Outcomes of Payne et al. [1993] and other DDM studies highlighted several important aspects which greatly affect rationality of decisions [Drury et al. 2011]:

- Decision makers perceive uncertainty differently.
- Information used for decision taking is limited.
- Behaviors are adjusted.
- Decision makers deal with complexity of problem differently, as well as with internal conflicts.
- Decision making should be considered in the context.

For example, when the uncertainty is involved, people are less rational and have to learn thinking systematically in the decisions involving risks, when clarity on benefits long-term is vague. When there are unknown factors, predictions with certain probability of occurrence can be made. This is a non-deterministic approach and produced options vary between decision makers depending on how they evaluate risks. As a result, because of uncertainty, multiple views are generated on the same situation. [Divekar et al. 2012]

3.3.3. Compensatory and non-compensatory decision making strategies

The researchers concluded that in the cases when decision making is concerned with multiple choices involving numerous attributes, decision model can be characterized as compensatory or non-compensatory [Payne 1976]. A compensatory decision making implies comparison of the variables values of each option in order to identify optimal alternative. In this principle of decision making high value of one variable can compensate lower value of another.

On the contrary to compensatory strategy, non-compensatory decision making implies shortcuts by simplifying variables evaluation. In this tactic selection of the best alternative is achieved through elimination of choices by variables threshold values. This makes the decision faster providing less efforts are needed to evaluate variables. The drawback is typically a resulting decrease in decision quality as not all available information is examined.

Decision makers tend to choose a decision making strategy based on many aspects, involving cognitive abilities, experience and information accessibility. The need to use more accurate strategies arises when the mistake in the decision has significant impact. In this situation inaccurate decision is not justified at the expense of saving time [Payne et al. 1993].

3.4. Decision making quality

Decision making quality is directly associated to the decision making process, which can be typically categorized as compensatory and non-compensatory. The compensatory decision making process is considered to be producing higher quality decisions. In the situations, where uncertainty is high or information is difficult to access / limited, the decision intelligence declines, thus leading to non-compensatory decision making process. [Payne 1976]

Drury et al. [2017] review of existing literature showed that some researches focus on relatively objective indicators of decision making quality, such as time and outcome, while others use extent of satisfaction with the decision as an evaluation criteria. Thus, researches are not consistent with use of either of decision quality indicators, and consequently, those may vary in the research of decision making in agile software development.

3.5. Decision making in agile context

DDM suggests decision making should be studied considering the context. This enables to reveal aspects influencing decision makers' behavior in addition to the ones already specified by DDM research, such as perception and amount of uncertainty, extent of information use, tendency to re-use old approaches / solutions, complexity of the problem and internal conflicts.

In the agile settings, there is an accuracy-effort trade-off between taking the time to generate and evaluate alternatives and time saving in favor of fast outcome. Agile teams' decision making process exhibits use of minimum information, reduced analysis and comparing of bad and good aspects of alternatives [McAvoy and Butler 2009]. This showcases the cutting-off in the considered variables in sake of easier way to derive the most important attributes. It means agile teams employ non-compensatory decision strategy [Drury et al. 2017].

Lipshitz and other researchers looked into characteristics of decision making in practice, and admitted that teams do not necessarily generate all possible options. Moreover, further comparing them may lack evaluation criteria and system [Lipshitz et al. 2001]. Overall, there is still little known about the obstacles agile teams encounter in the decision making [Drury et al. 2017].

Experience

Research revealed that "experience is a driving force in ASD project management decisions" [Drury et al. 2011].

Low team members' contribution

It was discovered that poor decision making can be a result of low contribution of team members in the discussions, in case they are not sharing opinions which are opposite to other team members [McAvoy and Butler 2009].

Limited information

Also, actual information is not being inquired and utilized at full in favor of saving time. Agile context creates situations for suboptimal decision processes such as groupthink, groupshift, or a liability to defer awkward decisions until the future [Briggs and Reinig 2010; Coyle et al. 2011].

4. Case study: Data Analysis

According to Simons [2009, p.21]:

"Case study is an in-depth exploration from multiple perspectives of the complexity and uniqueness of a particular project, policy, institution, program or system in a "real life" context. "

Considering importance of the context in agility evaluations, the empirical study is instrumental to identifying quality issues in the team's decision making and how outcomes of such decisions turn into quality issues in user stories and tasks, architecture design, delivery infrastructure support, code inspection and tests. While the case study focus is the agile usage in a single software development company, and generalisation of the results is not the purpose of the study, other agile teams may face similar issues.

4.1. Motivation and purpose of the case study

Considerable amount of existing research relates to positive factors of agile methodologies [Highsmith and Cockburn 2001], with some covering challenges when migrating to agile [Boehm 2002], yet there is little focus on the issues teams experience in the practice. Since agile process is empirical, as opposite to defined processes, there is no universal solutions but instead customized approach is required for each project case.

Collaboration is core in agile settings [Agile Alliance 2001]. This assumes not only communication between team members, but also ability to work together to achieve project goal [Highsmith and Cockburn 2001]. According to Highsmith and Cockburn [2001], the way how teams make decisions indicates the extent of collaboration.

DDM research summarised several principles as factors affecting decision making, namely uncertainty, limited information, behaviours which are adjusted, complexity of the problem or internal conflict in decision making, and contextual differences. Drury et al. investigated decision process in several agile teams, with the case studies which revealed issues like team members relying on others for decisions and team members not taking ownership, and collaborative decision making restraining experts opinions. [Drury et al. 2012].

While Drury and O'Dwyer studies are focused on agile planning and daily meetings [Drury and O`Dwyer 2013], the goal of this research was investigation in technical meetings. As it was previously noted, some agile teams tend to have solutions design in

smaller groups and out of agile planning meetings. The specific interest is the solutions that impact architecture or API design, systems scalability, as those impact agility of adding further changes to the system. The quality of technical meetings decisions impacts agility pace and quality of the resulted product.

The current study extends agile research by exploring agile team challenges with focus on people opinions which in practice define customised approach for applying agile in the project. The idea is to understand both perspectives, how they themselves identify issues individually, and at team level. It may also clarify why even at the time when team members have bright ideas, not everything is realized in practice.

Knowledge of the pitfalls experienced by agile teams is beneficial for the success in the agile usage. Moreover, revealing of such issues by teams on its own is hard as some issues relate to and require change of the team mindset. This increases value of awareness of possible obstacles that they may face.

4.2. Case study environment and the participants

A department of Information and Communication Technology (ICT) company developing a cloud-based product for the specific business case in the mobile telecoms industry participated in the cases study. The company has been delivering network solutions on the market over 10 years, and Capability Maturity Model Integration (CMMI) appraised for process improvements. The department is not dependent on other teams, and is utilising existing hardware for running the platform. While the company is a medium size, the project itself accounts for around 20 employees, consisting of developers and business development representatives.

The product in the concerned project is built using cloud microservices architecture. This provides quickly accessible cloud-based applications and enables to provide real-time services to any users around the globe. Each microservice can be changed independently without compromising the integrity of the system.

Since everything is dynamic in the digital world, the department leverages agile software development techniques to keep up with the demands of software evolution on the markets. A need for agile deliveries pushed the projects to move to microservices architecture and Kanban framework for software development.

The project exploits Kanban framework as a method for implementing agile software development. The team strongly focuses on the business case and MVP (Minimal Viable Product) approach when forming user stories backlog. This ensures

that only the set of features which are considered as bringing highest value are being developed. The project has been running over a year with business people involved and has recently gained attention of real customers. The survey was taken when the team was preparing for the first field trial to demonstrate the basic product features to potential customers.

The team is not under time pressure for the deliveries while the product is not in use as a service by the customers. Since the project is running without real customers involvement, the backlog priorities are also subject to the team decisions. The team is provided by high level buisness goals, and experiences hard times to define based on this input which functionality item to be implemented to achieve the goal and which represents greater value for the customer.

Majority of team members are experienced in software development, working as software developers over six years. The team accounts for 12 employees, with three persons joined within last two months (one newcomer did not participate in the questionnaire). The author of the thesis is a part of the team, and decided not to take part in the survey to avoid any interference with the results. So, overall 10 out of 12 developers answered the survey which accounts for 83% of the team.

The team is distributed with six people working in Tampere and others in another office in Finland. Therefore, the team ability to communicate effectively and agree on decisions is prominent to succeed with the project goals. Moreover, how the team leverages agile practices becomes even more important with the team growth and being at time of acquiring real customers.

4.3. Data collection methodology

A questionnaire was designed as a means to collect data for the identification and in-depth analysis of quality issues in agile software development. This method was considered as preferred over an interview, as it provides anonymity of answers and eliminates bias, otherwise existing in the interaction with interviewer. The questionnaire was reviewed by the thesis supervisors for its accuracy and validity to provide meaningful results. In order to reduce biased opinions when using multiple and single choice questions, the respondents were offered a choice to specify own option in addition to the suggested answers. Open ended questions aimed to collect elaborative information on the quality concerns of each team member.

The questionnaire offered multiple and single choice, rating and open ended questions. It started with an explanation of its purpose and security of data collected, highlighting also that the participation is voluntary and the person could stop the survey at any time if they do not want to proceed for any reason. The survey was divided into three main sections. Overall, there were 22 questions in the survey, which are enclosed in Appendix A. While designing the questionnaire, experienced people's opinion from both academy and industry were consulted.

The first section is general, and inquires team members experience in software development, agile projects, implementing microservices architecture and maintaining infrastructure supporting deliveries.

The second section aims at getting insight into a process of decision making in the team. This includes questions about size of the group taking decisions, information sharing, communication issues, satisfaction with the taken decision and decisions intelligence. The technical solutions design clarification is the desired outcome of technical decisions. The decisions quality in that case reflects technical solutions accuracy, and if there are shortcuts in the design, it would indicate increased likelihood of solutions removal. This part of the survey showcases also the challenges posed by inaccuracy of the agile manifesto in its formulation. Overall, the designed questions reveal the teams approach to implement the agile values in practice.

The concluding section examines team's readiness for fast deliveries. The questions cover user stories clarity and breaking into tasks, ability to solve infrastructure issues. Importantly, this gets insight in how team visions a need of improvements by enquiring team members opinion on the need of changes in microservices architecture, testing and code review practices. In order to identify quality issues, two approaches were used in the questionnaire. In the survey multiple choice and open ended questions, respectively, served this purpose. The designed multiple choice questions were targeting the deviation from the following chosen agile software development process quality goals:

- clear user stories,
- short implementation tasks,
- existence of testing to reveal architecture instability,
- team is prepared for infrastructure issues,
- brief testing exists to reveal major issues,
- team agrees with the benefit for the project from adding of code complexity and coverage measurements.

Further on, the designed open ended questions were identifying teams concerns, which reflect how developers themselves perceive the software development process quality, and its goals.

4.4. Data analysis

The collected information was interpreted and summarised using content analysis. The summary of the data was performed respective to each section of the questionnaire.

The responses related to the first section of the survey were analysed from the perspective of experience role in agile projects. The diagrams facilitated the research by visualisation of the results.

The second section of the questionnaire focused on the decision making process. The received data were grouped and matched to the agile values statements to determine how each of the agile values has transformed into team mindset in practice.

The last section of the survey served as a tool for getting insight into existing in the project quality issues in user stories, architecture, infrastructure, code inspection and tests. The answers were analysed for correlation with decision making process issues. The revealed impediments in the software development process, in its turn, are likely to slow down the software release.

5. Results

This chapter outlines the summary of the data collected with the use of the questionnaire.

5.1. Quality issues in decision making process

The found decision making process characteristics showcase the obstacles teams face when applying agile manifesto core values and principles in practice.

5.1.1. Decisions intelligence from experience perspective

The first section of the survey was enquiring background information on experience of team members.

Experience working in software development

Figure 1 illustrates that the developers are rather experienced in software development, bringing diverse background to the project.

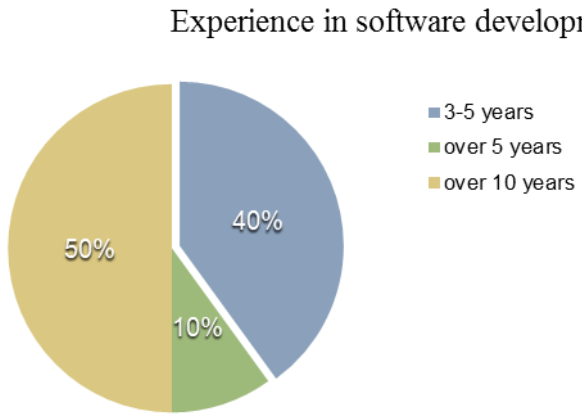

Figure 1 Software Development Experience

- *Experience working in Agile projects*

Over 50% of the team members have been working in the projects implementing agile software development longer than two years, the rest of the team had experience in agile projects for 1-2 years. This indicates the team is familiar with agile principles and applied them in at least one project.

Salo and Abrahamsson [2007] revealed that people who had experience working in agile projects viewed agile software development as useful, and Laanti et al. [2011] revealed a positive correlation between agile experience length and value of its usefulness.

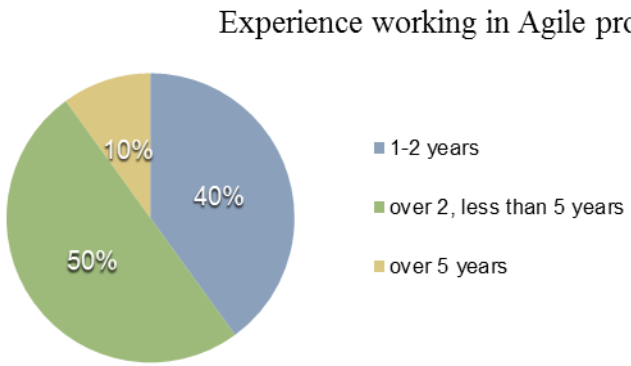

Figure 2 Agile Experience

Experience working with microservices architecture

Based on the gathered data, we can see that the team members are equally experienced in the microservices architecture, and the experience at full length is received working in the current project. The person who has around 4 years of previous experience in microservices has just joined the project. Since there was no experience of microservices architecture development from past projects, the knowledge about principles of microservices architecture design could be yet not fully available in the team. Having flaws in the microservices architecture, such as a lack of scalability and reliability, presents one of the risks for agile deliveries.

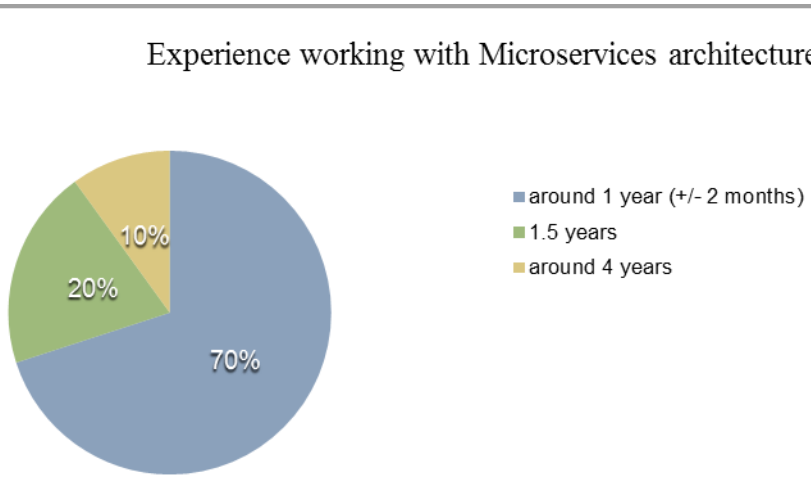

Figure 3 Experience in Microservices Architecture

In order to clarify quality of the developed by the team microservices architecture in the project, in the third section of the survey were questions about improvements that team suggests for microservices architecture.

5.1.2. Size of the group making decisions

70% of the team stated that technical decisions are taken in small groups of 3-4 people. On the contrary, the rest of the team considers the decision is discussed in the bigger groups, including more than 5 people. The setup of using small groups in this project means not everyone participates in the techical details of, for instance, user stories that other people are working on. In that case keeping up the rest of the team updated with the solution design decisions is important.

5.1.3. Information sharing

According to the answers, ongoing in the project technical discussions involve brainstorming and sharing the outcome with everyone on the team. One of the team members additionally specified that decisions are presented at daily or other team meetings, but the reasoning explaining the decision is not shared.

Moreover, regarding sharing of why the solution was chosen, 60% of the participants stated that the sharing of solutions pros and cons happens only *among the decision makers*. In contrast, 20% claimed that such details, explaining the outcome, are shared with everyone on the team. The remaining 20% of the team did not provide their opinion on this survey's question. It should be noted that insufficient information sharing reduces decisions intelligence due to lack of shared knowledge.

5.1.4. Opinions contribution

60% of the team answered that their opinions are not taken in the decisions making process, referring to having no opinion on the problem case (40%), ignored opinion (10%) and not willing to share with others own point of view (10%).

I do not have my own opinion on the problem case	My opinion is ignored	I do not want to share my opinion	Other
40%	10%	10%	40%

Table 1 Opinions sharing approach in the team (% of the team)

The rest of the team (40%), in this question of revealing impediments in sharing opinion, chose option "other", adding following comments:

- "Big team, lots of participants"
- "I'm afraid feelings are hurt"

- “I’ve learned not to participate much and go along with the flow, since the "key persons" in the project decide stuff that I make anyway. But sometimes I like that my opinion is heard and valued as well, yes.”

It means that decisions are usually taken considering only 30-40% of team members opinions.

5.1.5. Planning (solution design) can be skipped?

The team provided their perspective on the agreement with the statement “Planning (solutions design) can be skipped, best to start implementation as soon as possible, then redesign solution if needed”.

30% of the team opted for implementation start early rather than spending time on planning / solutions design. The remaining 70% disagree with this approach, reflecting that planning is important before implementation start.

If some of these 30% are the actual key persons in the team, it can significantly impact the decision making causing tendency to make solutions of insufficient design, and, consequently, these become temporary solutions which are subject to be removed or redesigned later on.

5.1.6. Decision making process organisation

Half of the team highlighted lack of structure in the discussions as one of the impediments in decision making. This may be one of the reasons making opinion sharing dificult and why the team does not understand why the chosen solution is better than other alternatives. Other reasons are specified in the Table 2.

5.1.7. Satisfaction with the taken decision

60% of the team highlighted lack of understanding why the chosen solution is better than alternatives. Other issues are summarised in the Table 2.

lack of structure in our discussions	lack of technical argumentation / knowledge of how to solve such technical issue	personalities of the decision makers rather than technical grounds	Not enough project history to answer
50%	30%	20%	10%

Table 2 Issues in the technical discussions

Lack of technical argumentation and influence of the personalities, lack of structure in the discussions are among the reasons why the team does not understand the reasoning behind the decisions made.

Technical discussions suffer from insufficient organisation and other mentioned problems, leaving benefits of solutions not clear for the participants. This indicates presence of insufficient rationality in comparing of solutions during the solution selection phase of decision making process, and possible implications in the solutions development phase generating a set of alternatives.

5.1.8. Understanding of Minimal Viable Product solution

There are numerous use cases that the product might support, the challenging task for the team is to identify which are needed to satisfy first potential customer. This means there is a high uncertainty of needed use cases for the particular features. The team handles this situation by highly utilizing delivering of MVP (Minimal Viable Product) solutions. This approach is used to maximize value of the product while avoiding implementation of use cases that are not yet confirmed to be needed. The term is broadly used in the team, and the questionnaire was seeking the information on which properties the solution should exhibit to be called as an MVP solution.

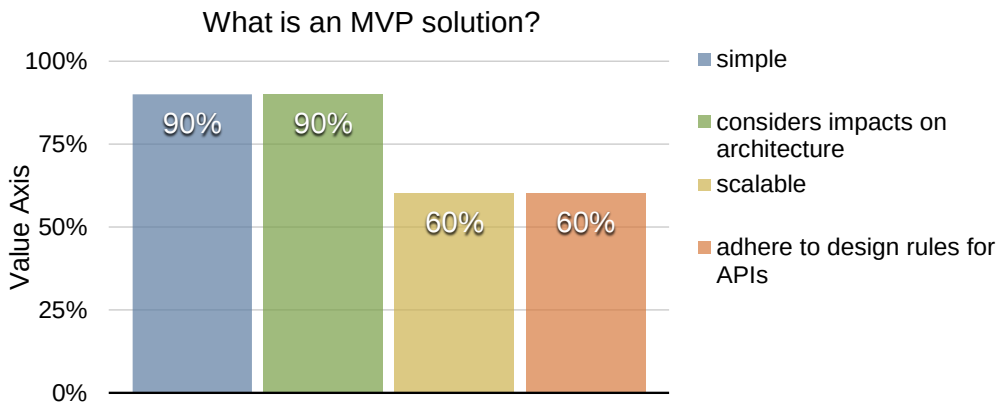

Table 3 Necessary properties of an MVP solution

As to the results of questionnaire, the team consensus is that MVP solution should be the solution which is simple and considers impacts on the architecture. However, 40% of the team stated that scalability or adherence to API design rules are not necessary attributes of MVP solution. This exemplifies that some MVP solutions could

generate temporary solutions and thus create the technical debt because of such solutions are limited in scalability or deviating from API design rules.

5.1.9. Temporary solutions

The team observes rework / removal of solutions in their daily work. 70% of the team members answered that they could anticipate the solutions removal from the beginning. This indicates that some decisions were fast intentionally, with the focus on rather temporary than long term solution.

Temporary solutions are signifiers of technical debt and extra work. Since the team does not have time pressure, this approach is doubtful in its benefits as not justified by current business needs and additional work could be avoided by putting more efforts in the solution's design. Moreover, achieving design excellence, promoted in the agile manifesto, does not seem to be efficient with the chosen approach of temporary solutions if such solutions are not accurate in architecture design.

5.1.10. Team concerns on decision making

The participants were asked to answer an open question seeking to reveal aspects of current decision making process which team classified as concerns or issues. The coding was performed based on the given answers, with explanation provided in the table below.

The team highlighted difficulty in sharing opinion (big team), hindered improvements (repeatitive decisions), lack of customer support (unclear priorities on use cases needed for the customer), too fast decisions and misunderstanding of MVP. As to collaboration issues, they complained that only few make decisions, and suggested that may be this is also the reason why others rather follow decisions than contribute to those.

Coding	**Clarification based on the received answers for the question "What are your major concerns about taking decisions process in the project?"**
Difficulty in sharing decisions among everyone	In large team communication is complicated by the need to share and align opinions with more people.

Coding	Clarification based on the received answers for the question "What are your major concerns about taking decisions process in the project?"
Improvements are hindered	Some decisions are not compensatory, actual factors are not considered, instead the decision is taken because similar was taken before. Being in the cycle of repetitive decisions, it is hard to change and improve.
Lack of end user/customer needs/requirements.	Decision priorities is difficult in the given situation when the knowledge of the customer needs is missing in the project.
Only few make decisions	Only few make decisions
Only few make decisions Not participate, follow others	As only few will make decisions, some make conclusion not to be involved much in the discussions, but rather just follow others.
MVP is misunderstood Many temporary solutions	Some team members are in favour of temporary solutions to deliver as MVP, aiming at saving a few hours now. Alternative would be to spend more time on finding the better solution.
Too quick decisions Many temporary solutions	Decisions are fast, it could be the reason for so many solutions which are only valid for couple of weeks.

Table 4 Explanation of the team concerns on the existing decision making process

5.2. Quality issues as an outcome of decision making process

5.2.1. User Stories and breaking the user stories into tasks

Quality Goal: Clear user stories

70% of team members admitted that the user stories are not clearly formulated.

Quality Goal: Short implementation tasks

According to the majority of the team (60%), typically, breaking user stories into tasks, lasts 2-3 days. 20% of the team reported the length of 1 day and the rest 20% specified that tasks take longer than 3 days.

Team concerns on quality in user stories and breaking the user stories down to tasks

As to the developers perspective on user stories quality, they defined deviation from quality goals in terms of following issues:

- User stories tend to be developers-oriented rather than user-oriented. Unclear user stories value; unclear what the user wants; missing end-user feedback; unclear use cases.

- User stories are not clearly specified with some being very large.
- Only few are involved in the user stories creation, thus others lack understanding of such user stories.
- Some work is not visualized by user stories / tasks.
- Some user stories are not correct due to suffer from missing / incomplete vision on long term solution for the architecture development.

Coding	**Answers for the question "What are your major concerns about the current process of user stories creation?"**
Less user-oriented	"They tend to be more developer / maintainter-centric than user-centric."
Unclear user stories value for end user	"The value created for end-user is not thought through and the backlog does not clearly visualize/communicate that."
Unclear what user story should be Missing end user feedback	"Nobody really knows what they should be. We are just guessing. MVP approach is dependent on continuous customer feedback, which we don't have. "
Poor specification of user story Work is not fully presented by stories / tasks Only few are involved in user story creation	"1) A small group creates a story, with a few words on a \\\"lappu\\\". Those that did not participate in the creation don't know what it's about. 2) Not all work done is represented by stories and tasks. "
Only few are involved in user story creation. User stories could be better	"Not everyone participates. We need more people to participate in the act of creating a user story. The more help we have creating these stories, the better they become. "
Unclear use cases	"use case is sometimes unclear"
Lack of understanding what user story tries to solve	"user stories are created without understanding the underlying problem. "
Missing end user feedback	"We don't have real feedback from the end users, so sometimes it's hard to groom. "

Coding	Answers for the question "What are your major concerns about the current process of user stories creation?"
Unclear architecture Some user stories are wrong	"Architecture's big picture is not clear, so sometimes wrong user stories are started to implement. "
Poor specification of user story Big user stories / tasks	"US too vague and therefore often also tasks too vague. Smaller US & tasks would be more straightforward to execute on Kanban from left to right within fewer days. "

Table 5 Team concerns on the user stories creation

Based on the received team's insight in the user stories quality, it is evident that the decision making process producing this user stories holds numerous issues. Among the issues, the uncertainty in views on customers' values and architecture long-term goals, which complicate the decision making process as every decision maker perceives uncertainties differently. Moreover, only few team members participate in the process of creation, indicating lack of information used for decision making.

Consequently, ambiguity in user stories formulation and scope, as well as large size, could complicate process of breaking down into tasks. This connection became evident also from the tasks creation issues which the team outlined in the further survey answers (see the table below).

The following issues could be specifically highlighted:

- Very large tasks.
- Lack of clarity in the formulated tasks.
- Not all tasks are created as soon as the user story is taken for the implementation in the iteration, even for simple user stories. Instead, tasks are rendered continuosly during the whole software development cycle.
- Difficulty to identify tasks needed to complete the user story.
- Lack of user story scope understanding (not clearly formulated user story needs).

Some team members assumed large user stories to be a potential reason for the observed issues with defining tasks.

Coding	Answers for the question “What are your major concerns about the current process of breaking down user stories into tasks?”
some tasks are too big	”Works quite well, some of the tasks are still too big”
some user stories tasks are discovered late, during development User stories are not initially well thought	”The backlog of the user stories are not initially thought - Tasks generate more tasks, even on relatively trivial user stories where things could've been anticipated. ”
some tasks are too big	”Some tasks are too big”
some user stories tasks are discovered late, during development User stories are not initially well thought	”It's not done early enough. Too often a story is started then it is split or tasks are added as the story continues. Ideally, a lot of this thinking would be done in the beginning. Then it is easier to track progress and helps to fully understand what task define the story to be done. ”
Unclear tasks	”sometimes tasks are not clear”
Lack of understanding what tasks are needed	”hard to know beforehand what tasks are needed”
Lack of understanding what tasks are needed Too big user stories	”Sometimes user story might be to big so that it is quite difficult to find all tasks.”
Lack of understanding US requirements	”Lack of knowledge of US requirements (what all is included and/or should be covered now). ”

Table 6 Team concerns on the user stories breaking down into tasks

5.2.2. Microservices Architecture

Quality goal: Architecture should be covered with performance tests

20% of the team stated that even in the present time there is no such tests. The rest of the team (70%) disagree that existing performance tests are sufficient, and one person answered that there is no need in adding performance tests. This means that team members realise a need to add performance tests to check how the chosen architecture is responsive and stable under a particular workload, how it is scalable and reliable.

This improvement if not put into practice may cause the need of architecture change, which may be complicated to implement when done at late stage of the project.

Team concerns on quality

The team outlined many issues, technical and team work related. The overall results can be found in the table below. The brief issues summary is as follows:

- Just few people know about metrics for the architecture, making it difficult to use them.
- Repeating similar decisions in REST APIs, not optimal ones.
- Not achieving architecture excellence: architecture is not abstract enough, code base is big and solutions value is less than efforts spent on adding them.
- Only basic scenarios are covered in tests: no practical evidence on architecture scalability, on how the system works in other use cases than "happy" ones.
- Lack of clarity on architecture goals.

It must be noted that some team members reported no issues, adding that there is no need of action becasue of short term goals are achieved, example of this is that tests are sufficient to run the system in basic scenarios.

Coding	**Answers for the question "What are your major concerns about the current architecture of the system?"**
Time to implement vs. benefit	"Microservices introduces a lot of overhead, which we are only benefiting of after 12 months and now that the team is larger = work can be separated logically. Also, REST APIs are time-consuming to create. Seems like we are repeating ourselves in a lot of places, when the actual use cases are relatively simple"
Not abstract enough	"It is not abstract enough."
metrics would be useful	"We could always have more tests. Metrics in our case would be even more useful but there are only a few people who have been involved in making these."
not tested how it scales	"not tested how it scales to tens of thousands"
Code base size vs. efficiency	"that relatively little stuff is happening, but yet the code base is already huge and inefficient"

Coding	Answers for the question "What are your major concerns about the current architecture of the system?"
Enough tests for running at the core	"Enough tests to keep the system running at the core."
Not clear view on architecture Microservices are handy	"We don't have clear view of architecture, but microservices are very handy."
Only "happy day" cases are covered No knowledge how the system behaves in mass disconnecting/reconnecting devices, big amounts of data, etc. No experience how to handle issues in production without impact on users	"Most of design is the happy path. NOK cases not that much covered? Also mass testing and abnormalities not that well tested (mass of disconnecting/reconnecting devices or a lot of api-oracle-queries when also devices report lot of data, can kuha-oss services handle it)? Also K8S & AWS configs/properties not that clear (how designed to be now and how allowed to change by e.g. Arska when incident happens on production env)!"

Table 7 Team concerns on the current system architecture

5.2.3. Infrastructure

Goal: Delivery Infrastructure knowledge is shared

60% of the team evaluated their ability of dealing with infrastructure issues as moderate / average, and 40% rated their skills as high / very high with distribution as to the table below.

0 (Very Low)	1	2	3	4	5 (Very High)
0	0	30%	30%	30%	10%

Table 8 Degree of ability of dealing with infrastructure issues within the team

In average, 90% of the team is working with cloud delivery infrastructure, including development and production environments, around 2 days per months. In contrast, one person in the project spends over 10 days per month on the delivery infrastructure issues handling.

"0-1"	"2-4"	"5-7"	"8-10"	"more than 10"
30%	60%	0	0	10%

Table 9 Amount of days per month spent on dealing with infrastructure issues within the team

The results indicate that infrastructure knowledge distribution in the team is not following shared knowledge approach, but instead it is concentrated in one person spending on which spends over 10 days per month on dealing with infrastructure issues and rated his skills as very high. The rest of the team attempts to catch up but two days per month is not enough to master the skill.

Team concerns on infrastructure quality

The overall results can be found in the table below. The brief summary of knowledge related issues is as follows:

- Fragmented knowledge.
- Not steep learning curve.
- Missing documentation.
- Changes are not traceable.
- Solving issues may be time consuming.
- Fragile infrastructure, may not be ready for big data use cases.
- The knowledge how to approach infrastracture issues is concentrated, rather than shared between team members.

Coding	**Answers for the question "What are your major concerns about infrastructure?"**
Fragmented knowledge Not steep learning curve	"Fragmented knowledge, high threshold to test/learn\"
Security of data in the cloud	"The security aspect of it. Everything uses encryption (or should), but the authentication and authorization is really loose in our infra. Once we get to production and big bucks are coming in, this is definitely something we should improve on"
No documentation No changes control	"It is a gang bang without documentation or any central control(like PRs"

Coding	Answers for the question “What are your major concerns about infrastructure?”
Solving issues may be time consuming	“Resolving problems might take a lot of time.”
The knowledge is concentrated in a few persons	“The knowledge of the infrastructure is held in the heads of too few people.”
Infrastructure is easy to break	“lab infra is fragile”
The knowledge is concentrated in a few persons	“That not many people know how to approach an infrastructure problem and how to even fix it.”
Infrastructure may not be ready to support big data scenarios	"Stability vs. constants changes in contents (and later when more customer device data involved), can we maintain it and keep data in DBs safe etc?\

Table 10 Team concerns on infrastructure quality

5.2.4. Code inspection and tests

Goal: Brief testing to reveal major issues fast

The team opinion on the need of extension of smoke tests has evenly divided (50% / 50%). It must be noted also, that how to extend the tests produced lots of alternatives, which however may not be realised if not supported by others in the team.

Suggestions for smoke test extension
“The critical user business cases that cannot break or else makes us lose big bucks”
“Test the autoconnection, alarms and pm counters.”
“more tests, volte. more devices”
“We already have ideas on how to improve it, we just need to make ourselves do it”
“There should be more end-to-end tests”

Table 11 Suggestion for brief testing extension

Goal: Measure code coverage and complexity

Code which has low coverage by tests or high complexity exhibits higher risk of

defects. Thus, by measuring these code metrics would allow to reveal code areas which change may be difficult. This approach could also be used as one of quality attributes benchmarks for each code delivery.

It can be seen that 50% of the team supported idea of automated continuous integration build extension with measurement of code coverage by test. In contrast, the rest 50% of the team evaluates it as not valuable.

The same statistical distribution was observed regarding add of code complexity measurements.

Team concerns on quality

The team suggested integration testing, code review and unit testing as a test and inspection activity that needs improvement first of all. In the table below can be found the arguments the team pointed out while explaining their choice of preference.

code review	**unit testing**	**System / integration testing**
30%	20%	40%
“too much focus on small details“ “sometimes code reviews are very lightweight“	“no UI unit tests and lack of UI testing knowledge“ “unit tests allow to look at the code from different angle. “ “Early faults detection. “	“Micro services can benefit from testing by metrics in staging environment.“ “Integration tests are fast in revealing unexpected changes that break the whole system. “

Table 12 Team view on code testing and inspection activities improvements

6. Discussion

6.1. Quality issues in decision making

6.1.1. Collaboration issues

Collaboration is a core team asset according to the agile manifesto. People, collaborating, can perform at considerably higher levels than when they work isolated. [Cockburn and Highsmith 2001]. Moreover, existence of expertise in the team in case of lack of collaborative planning could decrease team performance [Woolley et al. 2008]. The case study revealed numerous issues in collaboration.

Firstly, some team members were not participating and relying on others in making decisions. In this team many stated that they do not have opinion on the problem case, in spite of extensive experience in software development industry. This could be indicative of present in the team lack of accountability for decisions as "nobody being responsible" effect when all are responsible, with team members relying on opinions of more experienced team members [Drury et al. 2011]. From another side, this could be a consequence of insufficient knowledge sharing, when a developer does not have enough insight into the problem case because of not being informed on the previous aspects in the affected domain.

Secondly, there were as well people willing to contribute in the technical discussions, but whose opinion was not taken into account. Moreover, it must be highlighted that in this team work even experienced members had difficulty in communicating their opinion, stating that it is ignored by others. It may refer to the fact that some people are more aggressive or skilful in defending their point of view, in contrast to others who are avoiding conflicts and tend to agree with opposite opinion in favor of keeping peace. It is definite indicator that these people may further put even less efforts in defending their perspectives and eventually stop contributing as their opinions do not influence the outcome of the discussions. This may refer to one of the DDM principles which suggested that handling of ability to reflect on opposite opinions in case of internal conflict can influence the decision makers. Researchers suggested experience as being a driving force in decision making in agile settings. However, this case study also exhibits that even opinion of experienced people may be ignored.

As a result of the collaboration issues in decision making process, only opinions of 30%-40% of the team were considered in developing solutions. From the decision making quality perspective, lack of team members' contribution in discussions

decreases decisions intelligence, as the key information may be missing. Next, in order to operate as a collaborative team, and support intelligence of further decisions, knowledge sharing is critical. Since technical decisions were taken in small groups, the solutions were then communicated to other team members at other meetings, such as daily. There is no however the systematic approach used by the team, which would guarantee that all major technical decisions are communicated to the rest of the team.

In addition to opinions having no influence on the outcome, another negative experience that constrained team members creativity and may lead to their further isolation is the fact that the team while sharing the decision itself did not share motivation of the solution. Some agile teams have a practice of documenting solutions, or using whiteboards for following solutions pros and cons, this could possibly useful for this team to trace and, as well, improve the reasoning behind the options taken.

It must be noticed, that as to the practices used for holding technical discussions, the team members claimed that lack of structure in the decision making process was an issue for them. The improvement could be the environment which supports opinions equally, handling opinions clashes by rational reasoning rather than personal characteristics domination.

6.1.2. Short-term focus and lack of planning: temporary solutions

Another important insight gained from the case study was that the team focuses short term, while architecture related decisions require up front considerations as well. The team used MVP as one of the evaluation criteria when designing and choosing solution. Team members complained that there is no team consensus on this term regards to the limitations of the solutions to be delivered. Moreover, some developers highlighted that they could anticipate solutions removal. This indicates that the team uses shortcomings in the solutions in favor of being fast.

There is no time pressure for the deliveries in the project, but the team implements fast solutions as a part of agile approach chosen in this project. This phenomenon of making shortcomings with loose of big perspective on the architecture falls into agile anti-patterns when agility misunderstood with the team deviating from thriving for the technical excellence also mentioned in the agile manifesto principles. Beck and Boehm named design simplicity among the risks originated by agile approach [Boehm et al. 2005; Beck 2000], and this is evident in this team, and is leading to temporary solutions.

6.1.3. Difficulty in identification of the decision making issues

While the team holds retrospective meeting after each iteration, the issues in decision making and collaboration are hard to identify, instead, as DDM suggests, behaviors are adapted and decisions are repetitive based on the already made solutions / experience rather than looking for new approaches. Thus, team's mistakes agile manifesto interpretation are hard to reveal and address.

6.2. Quality issues as an outcome of decision making

The study identified significant quality issues the team observed in user stories and tasks, architecture and cloud delivery infrastructure. The findings reflect result of the corresponding team decision making in these areas. The quality issues were presented in the Data analysis chapter and further discussed with reference to decision making process and challenges to implement agile manifesto as most influencing factors.

6.2.1. User stories creation

In the studied team, user stories creation and breaking them into tasks is performed in small groups. This was also one of the issues, reported by the developers. Being isolated from the discussion, the team members were missing the reasoning behind the user story creation. Since the user story is poorly specified, the user story scope and value remain vague for the team members who have not been involved into the user story related forums. As a result of the knowledge not being shared and visualized well, misunderstanding of user stories increases, negatively impacting further decisions intelligence.

While user stories value and scope were not clear for those who was not involved in their creation, the decision makers were driven primarily by their own assumptions of what is needed for the customer. This resulted into user stories not being user oriented, but instead presented vision of this group of the team, which could differ from opinions of the rest of the team, and customer. The agile manifesto highlights importance of collaboration with business and customers. In this project the team business managers provided goals, letting the team to define how to reach this goal. In this situation uncertainty around value for the customer was high, which could lead to poor decisions on user stories regards to satisfaction of the end-users of the product.

6.2.2. Tasks generation

Tasks are created by breaking down the user stories. As a consequence of vague and large user stories, and consequent lack of understanding user stories scope and values,

the user stories breaking down into tasks was difficult and failed in producing complete set of tasks that would be required to accomplish user stories. This correlation was as well reflected in the team answers on the survey questions about tasks quality concerns. It exhibits that, eventually, issues from decision making on user stories populated in tasks generation.

6.2.3. Microservices architecture

It can be concluded that the team was oriented short term, both in architecture development and testing approaches, with lack of focus on architecture long-term design goals. The team highlighted that while the code base is big, it suffers from issues presented by not optimal and repeatitive solutions. The testing covers the basic tests, and some team members expressed their concerns regarding missing tests on system scalability, performance and negative scenarios. Majority of the team members agreed there is a need to add more performance tests.

It was noted that metrics could be helpful for arhcitecture quality measurement, however this knowledge was not available to the team member as not shared in the team. This indicates low level of knowledge sharing and collaborativeness in the team regarding metrics design and use.

6.2.4. Infrastracture

The knowledge of how to approach delivery infrastracture issues is concentrated, rather than shared between team members. The team members in average spend just two days per months on dealing with infrastructure issues, with one person being more often involved in it than others.

Since there is no documentation and changes control, the team has difficulty to grasp infrastructure and figure out how to approach arising issues. The agile manifesto suggests to minimise documentation, however it would be helpful for the team to have at least some troubleshooting guidelines documented to share this learning experience within the team.

6.3. Addressing decision making issues in agile software development

This chapter looks into approaches to embrace the challenges in decision making that the agile manifesto introduces for agile teams. Firstly, it is crucial to develop a tool for identification of decision making issues, which teams could use as a part of self-improvement in agile software devlopment. In order to build a framework for identification of decision making issues the paragidgm “Theory W“ [Boehm and Ross

1989] for the effective project management is leveraged. Secondly, software project management antipatterns are examined in order to provide solutions for the found decision making issues.

Antipatterns, when formulated as a pair problem-solution, becomes a tool for identification and addressing similar problems in the projects [Laplante and Neil 2006]. Thus, project management antipatterns, when matched to the found decision making issues, are instrumental in providing the basis for the solution in the context of the decision making. This approach was applied to collaboration and short-term focus / lack of planning issues independently, with summaries provided in the sections following the proposed solution for the decision making issues identification.

6.3.1. A framework for improving quality of decision making

In the case study it was revealed that self-improvement is hindered. By consulting project management antipatterns, agile teams difficulty in self-improvement can be explained by the project management antipatterns "Boiling frog" (behaviors are adapted), "The emperior's new clothes" (team might not know they are "naked emperior") and "One eye king" (when nobody challenges false claims) [Laplante and Neill 2006]. This chapter addresses the challenge in agile teams self-improvement concerning decision making process.

Quality of decisions can be evaluated based on the degree of satisfaction with the decision produced [Drury et al. 2017]. In this regard, a software project management theory "Theory W" [Boehm and Ross 1989], which aims at providing "negotiation with both parties feeling satisfied" and "the best interpersonal relationships" [Laplante and Neill 2006], can be used as a framework for the improvement of the decision making process. It promotes a win-win pinciple as essential for the effective project management. In its foundation there are following postulates [Boehm and Ross 1989]:

"*Identify win-win prerequisites.*
Construct a win-win process.
Build a win-win product. "

Therefore, by applying "Theory W" to the decision making process and leveraging the knowledge of the decision making issues, the resulting framework is expected to improve the quality of the decision making.

Identify win-win prerequisites for decision making process

This step consists of two major phases, which are 1) determining what everyone wants and agreeing on what is reasonable for everyone, and 2) establish an environment facilitating fulfillment of the revealed win-win requirements.

When applied to the context of decision making process, the *first phase* implies recognition of individual expectations of decision making process, and adjusting them to have everyone feeling satisfied with the resulting decision. In the ongoing project, this calls to understanding why the team members are not satisfied with the decisions made.

In the conducted case study, the team internal issues, such as collaboration obstacles and too fast decisions were contributing to the extent of satisfaction with the outcome. Consequently, the expectations at individual level but also enforcing collaboration of the whole team were derived, and presented below as a pair expectation and the decision making issue it was derived from.

Expectation 1

Everyone wants that his opinion is respected when making decisions.

Decision making issue: only few were deciding, others opinions were suppressed, with key persons dominating in the decisions.

Expectation 2

Everyone wants to be aware of the decisions made in the project and the reasoning behind.

Decision making issue: the outcome is not shared with everyone.

Expectation 3

Everyone wants to clarity why the solution is preferred over other choices.

Decision making issue: lack of technical argumentation explaining why the solution is better than others.

Expectation 4

Everyone wants structured discussion, which makes it simple to follow the ideas and contribute with own ones.

Decision making issue: Lack of structure in the decision making process.

Expectation 5

Everyone wants to spend enough time to elaborate solutions thoughtful from architecture and design perspectives.

Decision making issues: short-term focus and lack of planning, leading to temporary solutions.

The team participated in the case study did not complain about one of the known decision making issues, which is difficulty to make the decision. The list above could be complmented by the following expectation:

Expectation 5

Everyone wants that the decision is made in the reasonable time if there are several alternative options concurring for the best solution.

The *second phase* in the step of identification win-win prerequisites is setting up the environment, ensuring that defined expectations are fulfilled in the decision making process. The overall objective is to create a collaborative atmosphere where

- opinions are equally treated, and moreover,
- opinions are encouraged, thus motivating team members to share opinions.

It means, for example, in case of collision of points of view between them, the team should be equipped with practices which allow to develop and compare solutions with minimum impact from decision maker personalities. This would ideally exclude irrational domination factor in decision making process. As a result of collaboration issues, the produced decisions suffer from lack of intelligence and produce irrational outcomes.

In the ongoing project, setting up of the decision making environment calls to understanding of how to address the existing issues leading to the team members feeling not satisfied with the decisions made. In this context, the improvements suggested based on known antipatterns are useful.

- "Autonomous Collective" project management antipattern suggests review of the skills, and establishment of the leadership as essential for the decision making. One of the approaches to implement this, is that the team appoints someone ensuring that the decision making process fulfills the win-win prerequisites and leads to the decision with which everyone is satisfied. In this case, this person is responsible for the process, but making a decision is on the whole team. As the agile manifesto blends the roles in favor of knowledge sharing, the decision making responsible person could be different and changed from decision to decision. However, the leadership skills are important in this regard too.
- "Ant Colony" project management antipattern suggests identifying and eliminating the causes of fears restraining people from freely offering their opinions in the discussions. The reasons could be found in the behaviour of the most influencing people in the project, which is suppressing others. As a

solution, the antipattern suggests working with these people to introduce trust and encouragement in the team.

- "Fools rush in" project management antipattern emphasises importance of the planning in the solutions design before start using new tools.
- "Potemkin knowledge" project management antipattern highlights bad consequences of shortcomings are masked with fancy interfaces and quick results.

Construct a win-win process.

This step implies that teams set up systematic approach improving decision making. The main objective is monitoring the risks of deviation from win-win situation defined by the identified win-win prerequisites. This also implies the team detects and look into improvement for the discussions which are long, when when no common agreement from everyone in some technical discussions. By knowing the causes, the solutions could be developed, including deciding to start with one from the set of suggested best alternatives, by implementing a prototype which would further clarify uncertainties in the solution, researching more information on the problematic case, etc.. Agile retrospective meeting can be extended to implement this approach, with improvements for decision making process as an outcome.

Build a win-win product.

This step aims at aligning the process with the project needs. This can identify more factors influencing quality of decision making process, by specifying expectations on decisions as a resulting product.

6.3.2. Project management antipatterns for addressing collaboration issues in decision making

Within uncovered by the case study quality deviations, the collaboration issues are crucial. These have tremendous impact on team decisions quality, affecting both, the decisions intelligence and choosing an optimal variant, and also productivity - by discouraging people to contribute.

Project management antipattern "Autonomous Collective" [Laplante and Neill 2006]:

In order to become an autonomous team developing software, which is an essential property of an agile team in agile software development, requires broad range of administrative and technical skills. The anti-pattern asserts that it is erroneous to think that everyone in the team is equally skillfull and has needed abilities for performing varous tasks and making decisions. Thus, while an autonomous team is an appealing

approach for organisations in management of the projects, it is rather an eutopia rather than reality. In practice, instead of successful collaboration, the project may face lack of action and difficulties to achieve consensus in the team. Laplante et al. [2006, p.198] specifically underlines that "everyone cannot know the whole picture and everyone cannot be involved in every decision — not if we want decisions to ever be made." As a recovery from this situation, the teams are recommended to look into fostering leadership skills. Thus, it is important for the teams which implement agile methodology, to be aware of the skills needed and it is treendously critical to compensate "leadership vacuum".

Project management antipattern "Ant Colony" [Laplante and Neill 2006]:

In the foundation of collaboration, there is a principle of working together to achieve common goals. The anti-pattern claims that while at the surface it may seem that collaboration works well regards to this aspect, under the hood there are could be people which still disagree and silently adapt to what is desired from them. The reason of disagreement could be the conflict of the "common greater good" with personal advancements or lack of belief in the approaches chosen. Consequently, such team members feel discouraged to further engage in the decision making process. The behavior of adjusting instead of sharing concerns regarding the common goal and approaches, can be an effect of the team enviroment which does not support freedom of opinions, but instead feeds fears in minds of people when they try to contribute with point of view opposite to the what is suggested in the team. Conversly, the team atmosphere should facilitate trust and encouragement. As a solution, it is recommended to localise people and other factors restraining opinions contribution, causing discouragement and fears.

6.3.3. Project management antipatterns for addressing short-term focus and lack of planning issues

Excellence in the technical decisions has great importance on the quality of the resulting software product, including its maintainability, reliability and scalability to provide fast deliveries.

Project management antipattern "Fools rush in" [Laplante and Neill 2006]:

While it can be appealing to start using a new tool or methodology, it is erroneous to believe in its effective adoption by using an approach which neglects studying of the related implications. The planning is important in order to embrace the challenges coming with new endeavor. This implies that teams should allocate enough time to

research information on agile methodology, microservices architecture or any new gears, in order to benefit from using them. If the planning has already been limited or even skipped, the recommendation is, while in the rush, attempt to slow down things by bringing up questions on the solutions proposed and seeking the answers in the theory corresponding to the tool used.

Project management antipattern "Potemkin village" [Laplante and Neill 2006]:

While it is appealing to develop working functionality fast, it is erroneous to believe that quick solution is an effective approach. By implementation of shortcomings in the solutions and hiding defects, the real progress is masked by something that seems achieving the goal, but needs redesign or complete removal after the lack of quality or the fact of missing functionality is exposed. This may give wrong indication to the stakeholders on the work done and remained, and undermine the trustful relationships. Moreover, the system may loose integrity, and suffer from the issues found late in the project. So, as a result the short term benefits are likely to cause negative consequence on the project later on. The recommendation is to spend enough time and research on the solutions design, to implement scalable and reliable solution instead of the temporary one.

7. Conclusion and future work

The goal of this study was to understand the challenges agile manifesto presents for the developers and identify quality issues agile teams could observe in practice when using agile. Implications of the agile manifesto were outlined based on its critical analysis. Furthermore, agile team's decision making, as a crucial agile challenge, was investigated. The quality-centric survey was designed and one of the agile teams in Finland participated in this case study. The collected data was analyzed revealing the obstacles in agile manifesto application observed by this team, in terms of quality issues.

Agile methodology positions the team as a decision maker and collaboration as a key for the project success. However, the agile manifesto does not provide concrete solutions how to achieve collaboration and perform team decisions, instead, it is setting a mindset and goals, thus expecting that team is able to figure out how to interpret and achieve them. Moreover, agile values decrease the quality of decisions in favor of agility through reducing documentation and planning.

The case study showcases issues with achieving collaboration and making decisions in the technical meetings. Team decisions were dominated by key persons and developers were not sharing opinions. In addition to collaboration issues, team members did not understand why the selected options were better than others, and temporary solutions were chosen even that the team anticipated their removal, yet there was no time pressure in the project. Decision making issues populated into user stories / tasks creation, architecture, delivery infrastructure, testing and code inspection, resulting in bigger issues.

Interestingly, while agile methodology advocates continuous process and product improving [Agile Alliance 2001], this case study illustrates that collaboration and decision making issues are hard to reveal by teams as they are restrained by their mindset state and tend to stick to decisions from the past experience. Moreover, opinions given by developers in this research may not be indicative of further actions at team level because of the identified issues in the team decision making. This means some issues are not going to resolve as could be considered as not significant by the key team members. And by that same reason some of the issues may not be addressed in the proposed by the team members way.

While series of poor decisions present project failure risk, the team continued to make repetitive decisions, including mistakes, with retrospective meeting not being effective in revealing and solving them. The issues in agile team mindset are difficult to determine by the teams themselves, and require appropriate mechanisms. Thus, for agile teams to be able to identify obstacles in achieving agility while overcoming agile challenges, there is a need of mechanisms that would enable them to detect related issues.

This chapter is divided into three sections. The conclusions on decision making issues were summarized in the first subchapter. Then it proceeds with outlining of a proposal for framework equipping agile teams with techniques to identify the decision making issues. The framework was applied to the case study results and the recommendations were developed. The project management antipatterns were consulted to suggest improvements for the found in the case study decision process issues. The chapter concludes with describing limitations of the conducted research and proposal for future work.

7.1. Summary of decision making issues

The paper pursued getting insight into which issues teams may face in software development. With intention to understand which obstacles agile teams are exhibited to, agile manifesto was analyzed for challenges. Notably, it does not provide concrete solutions, but rather it is setting mindset and goals, expecting the team to be able to figure out how to implement them, for example to which extent minimize planning and documentation over working software.

Agile positions the team collaboration as a key for the project success. While agile mindset and practices serve the purpose of creating the cooperative atmosphere, it does not necessarily mean the team will achieve effective collaboration, where each team member contributes to decisions and shares knowledge. Poor decision making quality may become the reason of the project failure. The second section of the survey was specifically dedicated to study decision making in technical meetings which is one of the major agile challenges. The last section of the survey focused on the outcomes of decision making presented in user stories, tasks, architecture, infrastructure and testing.

According to the agile manifesto analysis, the teams collaboration success and decisions quality are directly affected by agile team mindset. As to the case study, teams can experience significant collaboration issues which affect decisions process and

outcomes on user stories, tasks, architecture, test and deliveries infrastructure matters. The major decision making issues were: only few were deciding, others opinions were suppressed, with key persons dominating in the decisions; the outcome was not shared with everyone; lack of technical argumentation explaining why the solution is better than others; lack of structure in the decision making process; short-term focus and lack of planning, leading to temporary solutions.

With minimization of planning and documentation, intelligence used to produce thoughtful decisions becomes limited. Moreover, as to the case study the agile manifesto transformed the team mindset shifting it to short term focus, rather than thinking of long term solutions, which are so important for architecture design and system scalability. In turn, shortcomings in solutions design reduce the quality of the team decisions.

Importantly, domain knowledge influences the quality of the decision as well. Since the team had no previous experience in the microservices architecture design, the decisions regarding microservices design could be especially difficult for the team. DDM highlighted uncertainty as one of the factors influencing decision makers.

In case of inaccurate decision, not optimal solution could be implemented. Temporary solutions as well as numerous issues in the team collaboration served as an evidence of this phenomenon taking place in the studied case. Decision making quality is instrumental for the excellence in the technical solutions design.

The research shows that the agile team saves efforts in finding solutions, and chose workable option rather than the best. This seems to work well in not-complex cases, however in the situations that require more thoughtful decisions reducing efforts in favor of saving time is arguable. Likewise, when decision making concerns design of technical solutions, lack of considerations may lead to temporary solutions, which may be solve particular use case but not consider architecture design, reusability and scalability. This may lead to the growth of technical debt, difficulty to introduce new changes and extra efforts to replace the solutions with long term. The real urgency in delivery may justify this approach, but having as a common practice presents the stated above risk may slow down further features development.

There is a difference between what agile manifesto dictates and how the agile principles are interpreted by the team, forming agile team mindset. Agile manifesto sets equations with compounds proportions which teams are expected to be able to figure out by themselves in order to achieve fast deliveries at high quality. At first, some

teams fail at collaboration, with some people putting forward their opinions better than others, thus leading to the effect when others are discouraged to share opinions keeping in mind previous bad experience of those not beings implemented anyway. At second, notably, the ideas, when suggested by the key persons are likely to take form of decisions in spite of opposite or different opinions in the team. For example, ideas of cutting of planning in sake of start implementing early. This is an anti-pattern in agile manifest interpretation as it leads to temporary solutions with loosing focus on the design excellence, even when there is no time pressure or customer that would benefit from this.

The study concludes that the decision making was nor collaborative, nor rational. As a result of opinions having no effect on the selection of options in decisions, people were less willing to contribute. Some people opinions were dominating over other members, even experienced. The team itself was not satisfied with decisions taken, referring to the lack of understanding of the selected solution motivation, leaving it unclear for the decision makers why it was better choice than alternative opinions.

Collaboration can be considered as a vehicle in distributing and maintaining knowledge in agile teams, which is essential in agile software development where documentation is minimized. However, lack of documentation was hindering the team work, as not well documented user stories, and missing troubleshooting guidelines for the infrastructure severely affected knowledge sharing and making decisions in this project. Siakas et al. [2005] pointed out that lack of systematic approach in agile software development can negatively impact the projects, and there is a need in further software development process modelling to integrate emerging thinking patterns, which also vary in different cultures and projects.

Software development is not just communicative, but collaborative. Such issues as developers, being not familiar with some decisions, doubting correctness and values of user stories, are indicative of lack of trust in the team. As to the studies, trust is inter-linked with software quality properties which span but not limited to correctness, consistency and reliability, and is built together with the product development through different stages of software development lifecycle [Berki et al., 2007]. The current research emphasized that the impact of human beings is first order for software quality.

7.2. A remedy perspective for emerging antipatterns in agile software development

In this study, it became evident that agile methodology possess a crucial gap. The agile manifesto presents great challenges by introducing the concept of collaborative and fast decision making, but yet there is no agile methods providing tools for agile teams to effectively self-improve concerning decision making. Firstly, there is a need in a framework which would provide means for agile teams to identify the decision making issues. Secondly, as soon as quality deviations in the decision making process has been identified, it becomes vital to address them appropriately.

Implications of the agile manifesto for decision making resulted in antipatterns in the agile usage, which further on manifested in decision making process. The proposed approach for addressing the decision making issues presents a remedy perspective for the emerging agile antipatterns.

A software project management paradigm "Theory W" is based on a win-win pinciple as essential for the effective project management. It is reasonable to assert that satisfaction with the decision produced is maximised when everyone conditions for the win-win situation are fulfulled. In addition, as to the literature reviewed, the extent of achieved satisfaction defines the quality of decisions. Thus, it was concluded that "Theory W" can be used for the decision making process quality improvement. As a result, the framework was derived for the effective decision making, which implies the following steps:

- Determining win-win preconditions: what everyone wants and agreeing on what is reasonable for everyone
- Establishing an environment facilitating fulfillment of the revealed win-win requirements.
- Setting up systematic approach improving decision making process. The main objective is monitoring the risks of deviation from win-win situation defined by win-win prerequisites set by the team. Agile retrospective meeting can be extended to implement this approach, with improvements for decision making process being an outcome.
- Specifying expectations on decisions in the project context, and incorporating the results into decision making process.

Based on the quality issues revealed in the case study, the following preconditions for the win-win situation were suggested for the studied agile team:

- Everyone wants that his opinion is respected when making decisions.
- Everyone wants to be aware of the decisions made in the project and the reasoning behind.
- Everyone wants clarity on why the solution is preferred over other choices.
- Everyone wants structured discussion, which makes it simple to follow the ideas and contribute with own ones.
- Everyone wants to spend enough time to elaborate solutions thoughtful from architecture and design perspectives.
- Everyone wants that the decision is made in the reasonable time if there are several alternative options concurring for the best solution.

The environment facilitating decision making process was suggested through addressing the existing issues which were causing the team members feeling not satisfied with the decisions made. The software project management antipatterns "Autonomous Collective", "Ant Colony", "Fools rush in" and "Potemkin knowledge" were consulted to propose solutions to deal with major collboration quality issues in collaboration and fast .

It must be highlighted, that the environment should implement freedom for sharing opinions, which would create trust and encourage people to contribute. As a result, increasing knowledge sharing in the project, generating more alternatives and improving likelihood of finding optimal solution.

When the environment is setup, the team should implement regular approach aimed at identifyig the deviations from win-win situation. One of the risks of deviating from the win-win situation is the decision making taking long time without producing the outcome. This risk should be monitored, and improvements made accordingly. The technical decisions excellence is critical for the agility, thus teams are suggested to spend enough time on the related decision making. The idea in addressing this risk is that teams should not jeopardise the solutions design phase in favor of being fast (especially the ones with impact the architecture and interfaces design), unless it is really forced by the current project needs.

7.3. Limitations and future work

This case study provides insight in the quality issues observed by a single agile team. Multiple case studies could generate more results, expanding the spectrum of obstacles teams face in agile usage. By covering longer period in the studied team, and involving

documentation, as well as project history review could deepen understanding of the causes of decision making issues. The research was performed in a small project having no dependencies to other departments in the developing of the cloud-based solution, it is reasonable to assume that results are relevant to small and medium enterprises, where the project is running with no big interference from outside.

Addressing found quality issues in decision making process could be advanced in further studies of software development antipatterns, in order to shape the solutions covering more cases of decision making issues. The effectiveness of the suggested framework in agile teams could be excercised, and further revised. Moreover, while the link to the agile manifesto was established for the revealed issues, thus showcasing presence of antipatterns in the agile usage as an underlying reason, other causes could be further researched.

In this research issues in agile manifesto implementation were revealed, and bad consequences outlined, with focus on collaboration and decision making. Issues related to misinterpretation of how to implement agility / agile manifesto, could be considered as antipatterns for agile usage. It can be beneficial to formulate them in the form of the framework used for project management antipatterns in innovation IT projects [Aydinli et al. 2016].

References

Acton, D., Kourie D., Watson B. Quality in software development: A pragmatic approach using metrics, *SACJ*, 52

Agile Alliance. 2001. Manifesto for Agile Software Development. Available as: *http://agilemanifesto.org/*. Checked November, 15, 2017.

Alleman, G. 2002. Agile Project Management Methods for IT Projects. In: Carayannis, E.G. and Kwak, Y.H. (eds.), *The Story of Managing Projects: A Global, Cross-Disciplinary Collection of Perspectives*. Greenwood Press, 1-22.

AlQaisi, R., Gray, E. and Steves, B. 2017. Software Systems Engineering: A Journey to Contemporary Agile and Beyond, Do People Matter? In: Marchbank, P., Ross, M. and Staples, G. (eds.), *Achieving Software Quality in Development and Use*. Southampton, 159-174.

Ambler, S. 2013. Scaling agile: the Software Development Context Framework, Available as: http://www.disciplinedagiledelivery.com/sdcf/. Checked November, 15, 2017.

Aydinli, D., Berki, E., Poranen T. and Stamelos, I. 2016. Management anti-patterns in IT innovation projects. In: *Proceedings of the 20th International Academic Mindtrek Conference*, Tampere, Finland, 1-10.

Barron, F.H. 1974. Behavioral decision theory: a topical bibliography for management scientists, *Interfaces* 5(1), 56-62.

Beck, K. 2000. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co, 2000.

Bell, D.E., Raiffa, H. and Tversky, A. 1988. *Decision Making: Descriptive, Normative And Prescriptive Interactions In Decision Making*. Cambridge University Press, 1988.

Berki, E., Siakas, K. and Georgiadou, E. 2007. Agile quality or depth of reasoning? Applicability vs. suitability with respect to Stakeholders' needs. In: Stamelos, I.G. and

Sfetsos, P. (eds.). *Agile Software Development Quality Assurance*. Idea Publishing, 23-55.

Berki, E., Isomäki, H. and Salminen, A. 2007. Quality and Trust Relationships in Software Development. In: *Proceedings of the SQM2007*.

Boehm, B. 2002. Get ready for agile methods, with care. *IEEE Software* 35(1), 64-69.

Boehm, B.W. and Ross, R. 1989. Theory-W Software Project Management Principles and Examples. *IEEE Transactions on Software Engineering* 15(7), 902–916.

Boehm, B. and Turner, R. 2005. Management Challenges to Implementing Agile Processes in Traditional Development Organizations. *IEEE Software 22(5)*, 30-39.

Buschmann, F. 2011. To Pay or Not to Pay Technical Debt. *IEEE Software* 28(6), 29–31.

Carleton, A., Herbsleb, J., Rozum, J., Siegel, J. and Zubrow, D. 1994. *Benefits of CMM-based software process improvement: Initial results. Technical Report*. CMU/SEI-94-TR-13.

Chow, T. and Cao, D. B. 2008. A survey study of critical success factors in agile software projects. *Journal of Systems and Software* 81, 961-971.

Cockburn, A. 2001. *Agile Software Development*. Addison Wesley, 2001.

Coffin, R. 2016. A tale of two projects [agile projects]. In: *Proceedings of the conference on AGILE*, 155-164.

Conboy, K. 2009. Agility from first principles: reconstructing the concept of agility in information systems development. *Information Systems Research* 20(3), 329-354.

Cunningham, W. 1992. The WyCash Portfolio Management System. In: *Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications*, OOPSLA, 29–30. ACM, New York.

Rajiv M. Dewan, Stephen C. Hansen. 1994. The role of user capability and incentives in group and individual decision support systems: An economics perspective. *Decision Support Systems* 12(1), 25-40.

Dillon, S. 1998. Descriptive decision making: comparing theory with practice. In: *33rd Annual Operational Research Society of New Zealand Conference*, 99-108, Auckland.

Divekar, A. A, Bangal, S. and Sumangala D. 2012. The Study of Prescriptive and Descriptive Models of Decision making. *International Journal of Advanced Research in Artificial Intelligence(IJARAI)* 1(1).

Drury, M. and O' Dwyer, O. 2013. An Investigation of the Decision-making Process in Agile Teams. *International Journal of Information Technology & Decision Making* 12(6), 1097-1120.

Drury-Grogan, M. L., Conboy, K. and Actonb, T. 2017. Examining decision characteristics & challenges for agile software development, *Journal of Systems and Software* 131, 248-265.

Drury, M. and O' Dwyer, O. 2011. Factors that influence the decision making process in agile project teams using scrum practices. In: *Proc. of the 6th International Research Workshop on IT Project Management.*

Drury, M., Conboy, K. and Power, K. 2012. Obstacles to Decision Making in Agile Software Development Teams. *The Journal of Systems and Software* 85(6), 1239-1254.

Duvall, P.M., Matyas, S. and Glover, A. 2007. *Continuous integration: Improving Software Quality and Reducing Risk*, Pearson Education.

Eloranta, V., Koskimies, K., Mikkonen, T. and Vuorinen, J. 2013. Scrum Anti-Patterns -- An Empirical Study. In: *Software Engineering Conference (APSEC), 2013 20th Asia-Pacific*.

Garvin, D. 1994. What does product quality really mean. *Sloan Management Review* 26(1), 25-34.

Hackman JR. 1987. The design of work teams. In: *Lorsch J Handbook of organizational behaviour*, Englewood Cliffs, NJ: Prentice-Hall.

Highsmith, J. and Cockburn, A. 2001. Agile software development: The people factor. *IEEE Computer* 34(9), 131-133.

IEEE Computer Society. 2004. IEEE standard for a software quality metrics methodology.

IEEE Computer Society. 1990. IEEE standard glossary of software engineering terminology.

Kasem, M. A. and Razali, R. 2017. Key Factors for Selecting an Agile Method: A Systematic Literature Review. *International Journal on Advanced Science, Engineering and Information Technology* 7(2), 526-537.

Klein, G. 2015. A naturalistic decision making perspective on studying intuitive decision making. *Journal of Applied Research in Memory and Cognition* 4(3), 164-168.

Kral, J. and Zemlicka, M. 2007. The Most Important Service-Oriented Antipatterns. In: *International Conference on Software Engineering Advances (ICSEA 2007).*

Kuranuki, Y. and Hiranabe, K. 2004. AntiPractices: AntiPatterns for XP Practices. In: *Proceedings of the Agile Development Conference (ADC'04).*

Laanti, M., Salo, O. and Abrahamsson, P. 2011. Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. *Information and Software Technology* 53, 276-290.

Laplante, A.P. and C J. Neill. 2006. *Anti-patterns: Identification, Refactoring and Management.* Auerbach Publications.

Lipshitz, R., Klein, G., Orasanu, J. and Salas, E. 2001. Taking stock of naturalistic decision making. *Journal of Behavioral Decision Making* 14, 331-352.

McAvoy, J. and Butler, T. 2009. The role of project management in ineffective decision making within agile software development projects, *European Journal of Information Systems* 18, 372-383.

Misra, S. C., Kumar, V. and Kumar U. 2009. Identifying some important success factors in adopting agile software development practices, *Journal of Systems and Software*, vol. 82 (11), 1869-1890.

Morris, K. 2016. *Infrastructure as Code: Managing Servers in the Cloud*, O'Reilly Media.

Payne, J. 1976. Task complexity and contingent processing in decision making: an information search and protocol analysis. *Organizational Behavior and Human Performance* 16, 366-387.

Payne, J., Johnson, E. and Bettman, J. 1993. *The Adaptive Decision Maker*. Cambridge University Press.

Qasaimeh, M., Mehrfard, H. and A. Hamou-Lhadj. 2008. Comparing Agile Software Processes Based on the Software Development Project Requirements. In: *Computational, Intelligence for Modelling Control & Automation, International Conference*, 49-54.

Rathor, S., Batra, D., Xia, W. and Zhang, M. 2016. What Constitutes Software Development Agility? In: *22nd Americas Conference on Information Systems*, San Diego.

Salo, O. and Abrahamsson, P. 2007. An iterative improvement process for agile software development. *Software Process: Improvement and Practice* 12(1), 81-100.

Siakas, K., Georgiadou, E. and Berki, E. 2005. Agile methodologies and software process improvement. In: *Proc. Intl. Virtual Multi Conf. on Computer Science and Information Systems*.

Simon, H.A. 1979. Rational Decision Making in Business Organisations. *The American Economic Review* 69, 493-514.

Simons, H. 2009. Case Study Research in Practice. London: SAGE.

Spinellis, D. 2016. Being a DevOps developer, *IEEE Software*, 33(3), 4-5.

Stoica, M., Mircea, M. and Ghilic-Micu, B. 2013. Software Development: Agile vs. Traditional. *Informatica Economica* 17, 64-76.

Vinekar, V., Slinkman, C.G. and Nerur, S. 2006. Can agile and traditional systems development approaches co-exist? An ambidextrous view. *Information Systems Management* 23(3), 31-42.

Woolley AW, Gerbasi ME, Chabris CF, Kosslyn SM, Hackman JR. 2008. Bringing in the experts: How team composition and collaborative planning jointly shape analytic effectiveness. *Small Group Research* 39, 352-371.

Zhu, L., Bass, L. and Champlin-Scharff, G. 2016. DevOps and its practices. *IEEE Software* 33, 32-34.

Appendix A

Quality Improvements in Agile Development. Making deliveries fast.

Page 2 of 4

Part 2. Working approach selected by the team for decision making.

2. * How many team members in your project are involved in the meetings dedicated to taking system technical decisions?

- 1-2 persons
- 3-4 persons
- 5-6 persons
- more than 6 but not yet the whole team
- the whole team
- Other (describe)

3. * Typically, what is included in a technical decisions taking in your project?

- brainstorming
- recording meeting minutes
- sharing the solutions pros and cons among the decision makers only
- sharing the solutions pros and cons with everyone in the team
- sharing the outcome among the decision makers only
- sharing the outcome with everyone of the team
- Other (describe)

4. * Typically, what kind of impediments with sharing your own opinion you observe when participating in taking technical decisions in the project?

- My opinion is ignored
- I am afraid to share my opinion
- I do not want to share my opinion
- I do not have my own opinion on the problem case
- Other (describe)

5. * Typically, what kind of general problems you observe in taking technical decisions in the project? Decision taking is highly impacted by …

- personalities of the decision makers rather than technical grounds
- lack of structure in our discussions
- lack of technical argumentation / knowledge of how to solve such technical issue
- lack of understanding why the chosen solution is better than other discussed alternatives
- Other (describe)

6. * Lets assume that there is no time constraints for the solution implementation. In your opinion, which of the following properties should have a Minimum Viable Product (MVP) solution in that case? Solution solves current use case and is ...

- [ ] considers impacts on architecture
- [ ] simple
- [ ] scalable
- [ ] adhere to design rules for APIs
- [ ] Other (describe)

7. * Some of the implemented solutions are removed/redesigned in the project. Could you anticipate its removal in the most of the cases?

- ( ) Yes
- ( ) No

8. * To what extent you agree that there is a need to change architecture design principles in the project?

Strongly Agree	Agree	Disagree	Strongly disagree
( )	( )	( )	( )

9. * To what extent do you agree with the statement: “Planning (solutions design) can be skipped, best to start implementation as soon as possible, then redesign solution if needed.

Strongly Agree	Agree	Disagree	Strongly disagree
( )	( )	( )	( )

10. * What are your major concerns about taking decisions in the project?

Previous Next

Quality Improvements in Agile Development. Making deliveries fast.

Page 3 of 4

Part3. Making software deliveries fast and with quality

A. User stories and tasks

11. * In your opinion, are most of the user stories formulated clearly, including the reason why this user story is needed?

- Yes
- No
- Other (describe)

12. * What are your major concerns about the current process of user stories creation?

13. * Typically, after breaking down user stories into tasks, how small are the tasks?

- 1 day
- 2-3 days
- more than 3 days

14. * What are your major concerns about the current process of breaking down user stories into tasks?

B. Architecture

15. * To what extent you agree that there are enough tests in the project for finding architecture performance issues?

Strongly Agree	Agree	Disagree	Strongly disagree	None of the above, as we have no such tests
○	○	○	○	○

16. * What are your major concerns about the current architecture of the system?

Quality Improvements in Agile Development. Making deliveries fast.

Page 4 of 4

24. Thank you for taking our survey. Your survey is almost complete, please leave any comments on this questionnaire, then press the 'Submit' button.

Previous Complete

D. Testing and code inspection

20. * Would you like to change or extend existing smoke test?

- Yes
- No

21. * Based on your experience ...

which test / inspection activity in the project should be improved first of all?	Please explain your answer
○ code review ○ unit testing ○ integration testing ○ Other (describe)	

22. * To what extent do you agree the current project would benefit if Continuous Integration build is extended:

	Strongly Agree	Agree	Disagree	Strongly disagree	None of the above, as we have no such tests
with checks that test code coverage is at the same level or better after each delivery	○	○	○	○	○
with measurement of code complexity	○	○	○	○	○